Faculdade de Engenharia da Universidade do Porto

Conversational AI: Automated Visualization of Complex Analytic Answers from Bots

Bruno Pereira de Morais

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Major Automação

Supervisor: Henrique Lopes Cardoso

July, 2018

Resumo

Num mundo que recolhe mais informação do que nunca, a capacidade de compreender rapidamente grandes quantidades de dados de alta dimensionalidade é uma tarefa importante, mas complexa. Para enfrentar este desafio, a Critical Software está a desenvolver o *Ansa*, uma plataforma e assistente conversacional e de pesquisa. Uma das características mais importantes deste tipo de assistente para análise de dados é a sua capacidade de gerar automaticamente representações visuais perspicazes a partir da informação colecionada para dar resposta à pergunta do utilizador. Isto requer um sistema capaz de decidir que informação apresentar, qual o tipo de visualização adequado e como utilizar a informação da forma mais apropriada nos devidos parâmetros da visualização.

Para tal, este trabalho apresenta uma solução baseada em *case-based reasoning* (CBR). O CBR, ou raciocínio baseado em casos, pretende resolver novos problemas recorrendo a informação relativa à resolução de casos prévios, e adaptando a sua forma de resolução para a nova situação. Baseando-se em casos de sucesso verificados, o sistema de CBR desenvolvido consegue decidir como organizar uma representação visual de dados apropriada sem a necessidade de conhecimento aprofundado ou regras definidas. É também capaz de aprender através do feedback do utilizador e ajustar a sua base de conhecimento conformemente. No seu funcionamento, o sistema compara o novo problema, os dados para os quais deve gerar uma visualização, com os seus casos conhecidos, determinando qual deles é o mais semelhante. O caso selecionado é recuperado da base de casos e a sua solução (constituída pelo tipo de gráfico ideal e pelo uso apropriado dos atributos em eixos ou séries) é adaptada de forma a ser aplicada no problema atual. Sendo determinada a visualização ideal para o novo problema, o feedback do utilizador dita se esta deve ser adicionada à base de conhecimento ou se o sistema deve apresentar uma visualização diferente.

O sistema apresenta resultados promissores, com elevada precisão e desempenho. A sua capacidade de apresentar bons resultados, aprender por experiências passadas, adaptabilidade a futuros tipos de visualização e fácil manutenção motivam a continuação da experimentação com esta técnica.

Palavras-chave: *Case-based reasoning*, visualização automática, visualização de dados.

Abstract

In a world that collects more information than ever before, quickly making sense of complex high-dimensional data is an important but intricate task. To tackle this challenge, Critical Software is developing *Ansa*, a conversational and search assistant platform. One of the most important features of this kind of data analytics assistant is its ability to automatically generate insightful visual representations from the relevant data resulting from the user's query. This requires a system able to decide which information to display, which visualization type to use, and how to arrange the data in the most fitting parameters.

To solve this automatic visualization challenge, this work presents a case-based reasoning (CBR) approach. Case-based reasoning aims to solve new problems by recalling previously solved ones and adapting their solution. By relying on verified successful cases, the developed CBR system is able to decide appropriate visualizations for the input data without the need for data visualization expertise or specific rules. It is also able to learn from the user's feedback, adjusting its knowledge base accordingly. In its operation, the system compares the received data to its known cases to determine which is the most similar one. The selected case is then retrieved from the knowledge base and its solution (composed by the selected chart type and attribute use) is adapted in order to solve the current problem. Once a visualization arrangement is selected, user feedback dictates if it should be added to the knowledge base or if a different solution should be presented.

The system shows promising results, with great accuracy and performance. Its ability to provide good results, learn from previous experiences, adaptability to future visualization types and easy maintenance motivate further experimentation with this technique.

Keywords: Automatic visualization, case-based reasoning, data visualization.

Acknowledgements

Gostaria de agradecer ao Professor Henrique Lopes Cardoso e ao Eng. Paulo Gomes pela sua contribuição na realização deste documento. Agradeço também a toda a equipa do Ansa, não só pela aprendizagem e disponibilidade, mas também pela excelente integração que me proporcionou durante a realização desta dissertação.

Deixo também um grande agradecimento à minha família mais próxima, por sempre garantir que nada me faltava e que tinha ao meu dispor todas as ferramentas para o sucesso. Obrigado pela vossa presença, dedicação e também pela vossa paciência.

Quero também deixar um agradecimento especial à Sarinha, por todo o apoio, preocupação, amor e carinho, sempre. Obrigado por me fazeres tão sortudo.

Por fim, agradeço a todos os amigos que tornaram o meu percurso pela Faculdade de Engenharia um período inesquecível. Sejam Extraordinários, Clube das Minis, BEST Porto, camaradas de ataque aos exames ou simplesmente boa malta que tive o gosto de ir conhecendo. É também graças a vocês que tudo isto compensou.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
CBR	Case-Based Reasoning
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JS	JavaScript
NLP	Natural-Language Processing
RBS	Rule-Based System

Chapter 1

Introduction

We live in a time when data is generated at an unprecedented rate [1]. The amount of information generated in the business world is quickly becoming overwhelming to interpret, yet it is crucial for management decisions. Quickly making sense of such complex high-dimensional data, originating from numerous sources and presenting itself in various formats, is an important but intricate task. To tackle this challenge, Critical Software is developing *Ansa*, a conversational and search assistant platform intended to change the way companies interact with their data.

Virtual personal assistants are a common presence nowadays, providing a wide variety of services which increases by the day. However, some complex tasks are yet to be addressed by them, namely data analysis.

1.1 Motivation and Goals

The ambition behind Ansa is to develop a flexible solution, ready for adoption in different business areas, that allows its users to do analytical questions and queries in natural language for their data, such as “what are my top five customers in USA?”, “What were my TV sales in Japan last year?” or “Show me my sales volume throughout 2016”, and get the most appropriate response for their analysis. One of the important components in this kind of analytics assistant or chatbot is the way it shows its answers. Especially, the analytical answers that are best presented in a visual way, requiring clean, perceptive and adaptable graphical representations.

Selecting the most appropriate visualization to represent the data requested by the user presents itself as a vital challenge. Most people are not familiar with the graphical design principles needed to arrange data into graphical representations that support their reasoning process or communicate their analytical results to others [2]. As such, the system’s ability to automatically provide insightful visual representations is one of its most value-defining features. Thus, the goal of this dissertation is to create a system able to adapt itself to user’s preferences and types of data, and create such visualizations. It is also important that this system does not require complex maintenance and can be used in Ansa’s future when more visualization types are added.

1.2 Solution and Approach

To face this challenge, this dissertation presents a case-based reasoning (CBR) approach. Given the information retrieved by the user's query, the developed solution is able to decide which of Ansa's visualization types to use and how to arrange the data attributes in different axes and/or series. This system, inspired by human reasoning, uses information from previous experiences (called *cases*) to understand and solve new problems. After determining which of the known cases is most similar to the new problem, the system compares the features of both and, having matched the case's features to the problem's features, it adapts the case solution in order to solve the new problem. It is also able to learn from user feedback, generate different solutions and it is easily adaptable to new graphical solutions which might be implemented in Ansa's future.

Developing such a solution requires answering some important questions. It is necessary, for instance, to determine what constitutes a good visualization and what key information defines the problem. For the success of the system, stored cases should hold the most relevant problem features and avoid unnecessary information. It is also critical to determine how to properly compare the cases to each other, as well as how previous experiences can be adapted to solve new problems successfully.

To allow for free experimentation, this solution has been developed separately from Ansa's system, although access was granted to use its test databases, perform tests and analyse its back-end and front-end responses.

1.3 Structure

This document starts with an analysis of Ansa, in Chapter 2, explaining the typical uses of the system and the kinds of response it provides. There is also a section on rule-based systems, the methodology currently in use for selecting its visual representations.

Chapter 3 presents the background knowledge for the developed work, covering general topics related to visualization as well as presenting an overview on projects related to Ansa and its goals. It ends with a detailed study of case-based reasoning, the focus of this work, covering its cycle, each of the main steps as well as its advantages and pitfalls.

In Chapter 4 the proposed approach is presented. It starts by detailing the development methodology, followed by an example use case and the project requirements defined by the team. Afterwards the system's architecture is described, as well as the modules that compose it.

Chapter 5 presents the developed work, from its environment to the implementation of each of the main CBR factors and stages.

In Chapter 6, system tests are described, evaluating its performance and accuracy. These tests are followed by the results discussion, where the outcomes of each type of test are analysed.

Lastly, Chapter 7 presents a conclusion on the developed system as well as a comprehensive reflection on the following steps and experiments to take to improve the current implementation.

Chapter 2

Ansa Platform

In this chapter an overview of Ansa's system is presented, showing its user interface, functionalities and methodology for selecting visual responses.

2.1 System Operation

Ansa is a web-app, thus the user accesses it via web browser. After asking the user for login information, the current Ansa prototype reveals a clean page, highlighting its main tool, the search box, as seen in Figure 2.1.

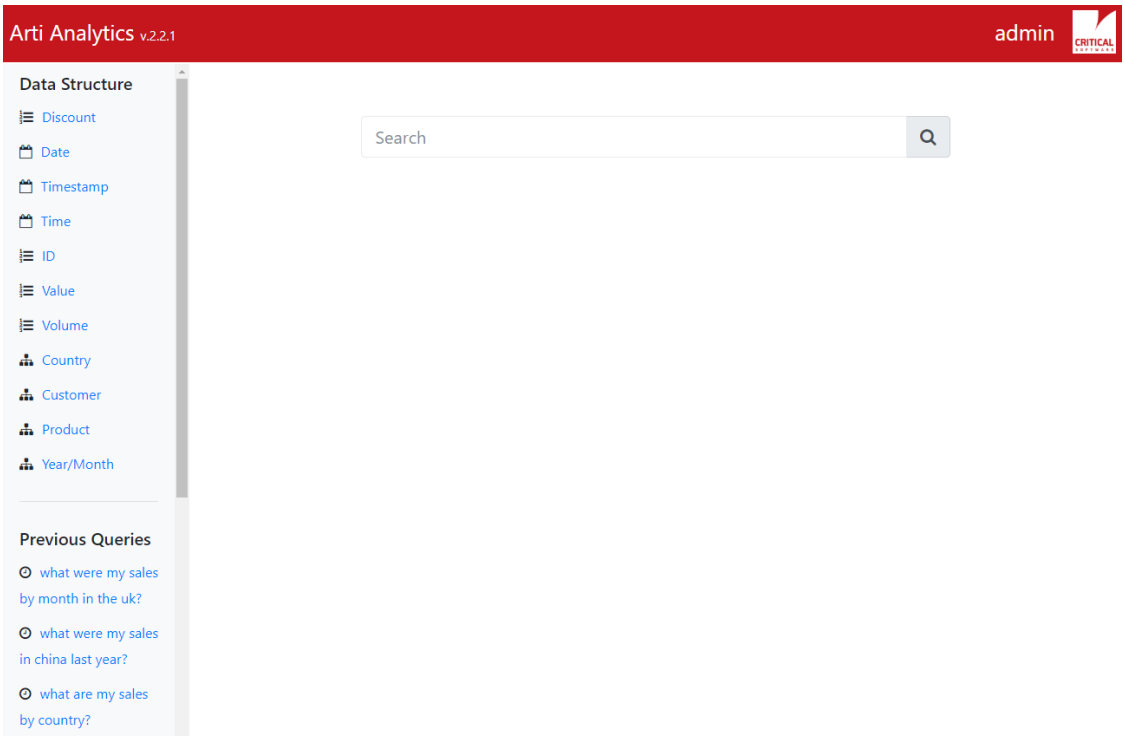


Figure 2.1 - Ansa Front Page

On the left panel it is possible to see the kind of information available in the database Ansa is running on. In this example a sales database is being used, which simulates the data of a generic business with products, customers and sales. Clicking each of these entities provides a detailed view of its content. This kind of result is presented as a simple table, as shows Figure 2.2. This “Data Structure” panel ensures the user keeps in mind the information he/she is working with, making it easier to create good queries for different kinds of analysis. Below it, the “Previous Queries” panel displays the history of previous queries input by the user.

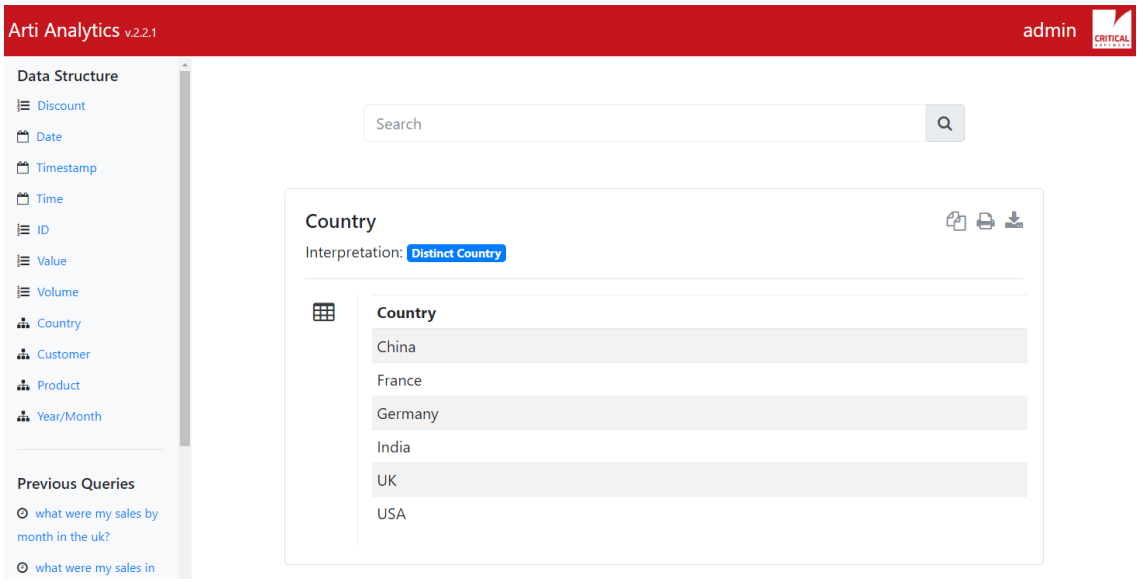


Figure 2.2 - Detailed View of "Country"

Ansa answers questions in different formats. In some cases, as seen in Figure 2.2, it returns a table to the user. For simple questions like “What is my total sales volume?”, the answer may be a single number (Figure 2.3), while more complex questions, such as “What were the sales by quarter by product in Germany in 2016?”, are best answered with a chart (Figure 2.4). For each query the system displays its interpretation, which is useful for understanding how it recognizes natural language and for debugging purposes. After retrieving the relevant data from the database, it is processed and fed to the visualization system, which determines the most relevant visualization to use. Whenever the system detects it is possible to display the information in other formats it includes the corresponding mock-ups on the left vertical bar. In the case of Figure 2.4, it is also possible to display the answer as a bar chart or in simple table format, yet the system displays the line chart first because it assumes it is the best representation for the query input.

With the purpose of having a better visualization of the platforms’ responses, the following output examples in this paper will contain only the answer box, ignoring the header and sidebar.

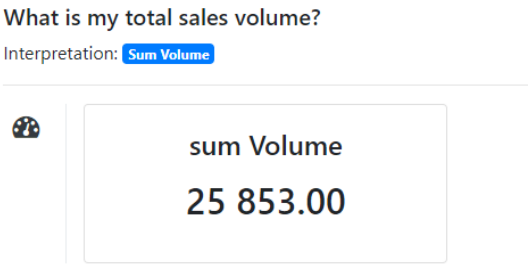


Figure 2.3 - "What is my total sales volume?" Output

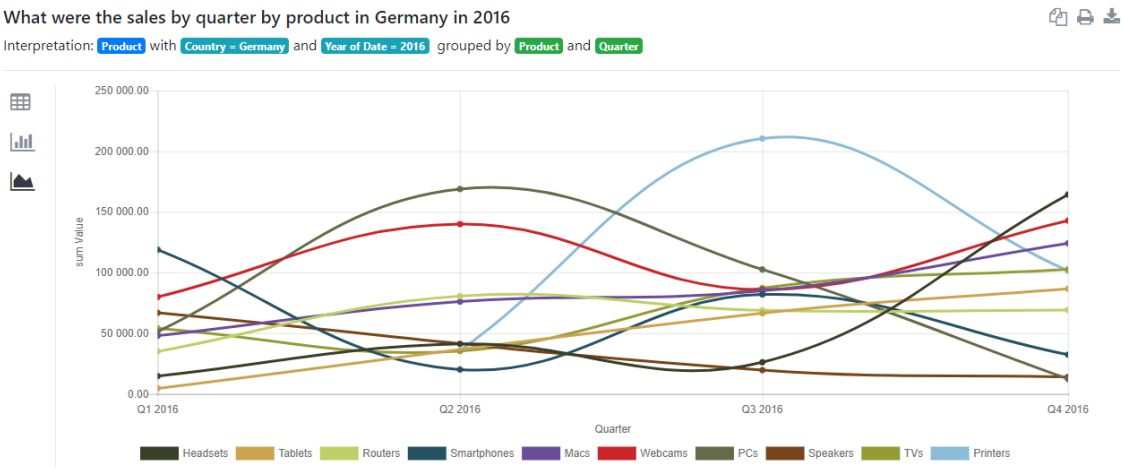


Figure 2.4 - "What were the sales by quarter by product in Germany in 2016?" Output

One of Ansa's development goals is to be able to provide the most insightful output for the user's query. Since most answers will be complex and the platform is expected to be usable in different databases without requiring a tailored set up, there must be a flexible solution for analysing the data and transforming it into the most useful kind of visual representation, with the most fitting parameters. Besides line charts (Figure 2.4) the other kinds of visualization currently implemented in Ansa are bar charts (Figure 2.5) and pie charts (Figure 2.6).

Independently of the visualization type, hovering over an area or data point of interest provides detailed information about it. In Figure 2.7 the mouse was hovering the Male section of the pie chart, revealing the total count of male employees.

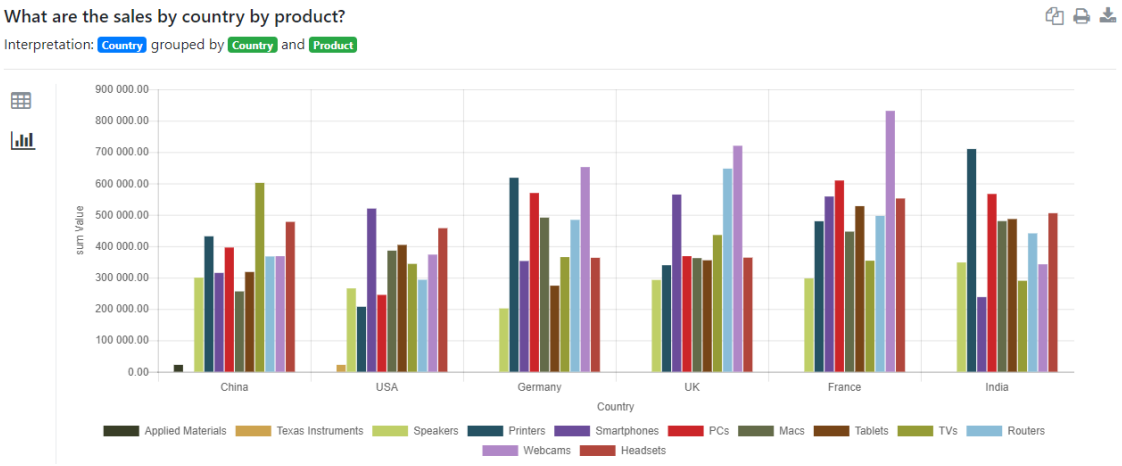


Figure 2.5 - Bar Chart Output Example

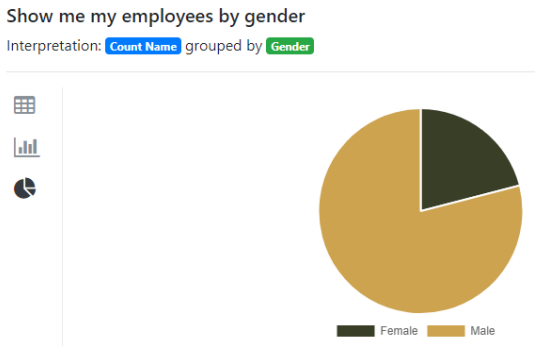


Figure 2.6 - Pie Chart Output Example

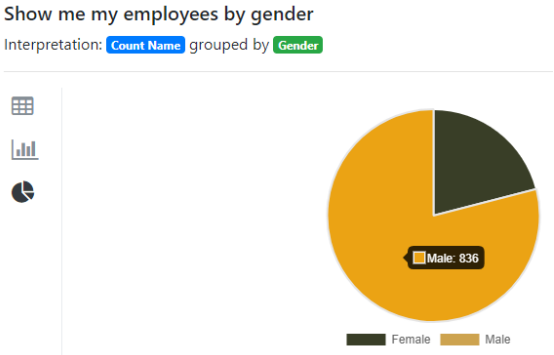


Figure 2.7 - Pie Chart with Mouse Hovering

2.2 Rule-Based System

The implementation of automatic visualization in Ansa is currently done using a rule-based system (RBS), meaning each chart representation shown by the system is the result of a set of rules put together for that specific kind of query outcome and chart type. Upon receiving and processing the query results, the different parameters must fall within one of the predesigned rules for chart selection and creation, where the corresponding actions are applied for determining which columns of the resulting table (referred to as *attributes)*, should be used as the X-axis, the Y-axis and the Series of the chart response. This process is described in greater detail in Section 4.4.2.

A typical advantage of rule-based systems is that, when the domain is narrow enough and well understood, experts are able to express their knowledge in the form of rules for problem solving [3]. In this case, by creating a substantial set of rules, which consist of a series of *IF-THEN* statements, it is possible to produce accurate. Rules are a natural form of expressing knowledge in a domain and these usually provide a uniform structure, where each rule is an independent piece of knowledge, resulting in a modular system.

This is the kind of approach currently used in Ansa for determining data visualizations. It is a common solution which has shown good results for other problems [2][4][5] and has proved to be extremely effective in the right conditions, using the right set of rules.

However, there are some inconveniences that must be considered [3][5]:

- **Infinite chaining**: Rule-based systems must be crafted carefully to avoid infinite loops.
- **Possibility of contradictions**: As the knowledge base (rules set) increases, its modification can get increasingly difficult. Introducing new knowledge to solve a specific problem (e.g., adding a new rule) may cause unexpected contradictions with the previous rules. Equally, if an existing rule is to be changed, the way it affects the remaining ones must be carefully studied to avoid unplanned consequences.
- **Opaque relations between rules**: Although individual rules are relatively simple and self-documented, their logical interactions within large rule sets may be hard to grasp, making it difficult to observe how individual rules serve the overall strategy of the system.
- **Complex domains**: Some domains are of such complexity that solving them with a RBS would require tens of thousands of rules (e.g., using a rule-based system to deal with air traffic control).
- **Inefficient search strategy**: For large rule sets efficiency is a challenge. Since the inference engine searches through all the rules during each cycle, this can result in a slow exhaustive process. This way, extensive rule-based systems can be unsuitable for real-time operations or applications where the user expects a fast response.
- **Inability to learn**: In general, rule-based systems do not have an ability to learn from experience. Unlike a human expert, who knows when to deviate from the rules or how to learn from a new successful or failed experience, an expert rule-based system can not automatically modify its knowledge base, or adjust existing rules or add new ones. The knowledge engineer is still responsible for manually revising and maintaining the system.
- **Dependency on domain expertise**: In order to provide good results there must be a very solid understanding of the domain in which the system is operating. Even if there is an extensive rule list, it is not enough to create a good result if the domain is not well understood or properly represented.

Some of these RBS characteristics are not desired for a system that intends to be a flexible tool adaptable to various kinds of users with different use cases and purposes, such as Ansa. Currently, the implemented rule-based system is capable of providing good visual representations for most types of data that it has been programmed to deal with, but the inability to learn means it may be dependable on adjustments for other kinds of data or visualization settings that may come up in the future, removing from the table what would be a very interesting feature to have in this kind of system. Moreover, the dependency on domain expertise is another disadvantage, as determining the most appropriate data visualization for all kinds of data is not a task for which there is a strong underlying understanding [6].

Motivated by these challenges, the Ansa team was set to experiment with a different approach. Previous experiences with case-based reasoning in other systems drove the team to try the technique for the problem of selecting the best data visualization and its parameters for a given data response. That decision resulted in the proposal of this dissertation.

Chapter 3

Background

The past few years have been marked by the rapid development of computer hardware and software [7]. This noticeable growth in computing power has enabled us to collect and store data at higher rates and scales than ever before. Often the information is of such magnitude that simply reading through it is not enough to understand it. In the words of S. J. Walker [8, p. 1], *"We're in an age where numbers no longer incite awe, since they've become loosened from the mooring of human scale and cognition. What does it mean when we are informed that Google produced more than 24 petabytes of data per day, or that the world sends 400 million tweets a day? Such numbers neither impress nor disappoint any more"*.

Being able to fully understand the information at one's disposal is a great challenge which can provide vital insights. It is often necessary to make crucial decisions based on collected data. To help us do so, the fast accumulation of data has incited the demand for developing strategies to display high volumes of data in a comprehensible way. To achieve it, many techniques were developed, collectively known as data visualization. A visualization does not have to include all the available information on what it intends to represent, but it must be based on it or its characteristics. What motivates data visualization is that it allows displaying large volumes of data in a compact, yet comprehensible manner. Data visualization as a practice can be traced back to early times, when man first learned how to draw [7]. Findings, such as some of the drawings in Lascaux's cave system, estimated to be 17,300 years old, show what is believed to be maps of stars and constellations [9]. Yet, throughout the course of history, data visualization has been an unpopular topic with few developments. In the seventeenth century, René Descartes contributed to it with a critical development, the invention of the Cartesian coordinate system, which allowed data to be visualized using two dimensions (or even more if points are given different sizes, colours, symbols, etc.), revolutionizing mathematics at its time. At the end of the eighteenth century, modern signs of data visualization became apparent as people started using the Cartesian coordinate system to display line graphs.

Over the last decades, developments in informatics have caused a rapid growth in interest and progress of the data visualization field, along with the development of better hardware and software able to render complex imagery. Today, data visualization is considered to be a branch of modern descriptive statistics, involving the development and study of techniques that convert raw data into imagery [7].

In the following sections, important aspects of data visualization will be discussed. This is followed by an analysis of projects related to Ansa and their approaches, as well as an in-depth study of case-based reasoning, this thesis' technique of focus.

3.1 Visualization

Throughout the years the meaning of *visualization* has shifted. According to the Shorter Oxford English Dictionary [10] from 1972, it used to mean *constructing a visual image in the mind*, while nowadays this is a secondary meaning, giving way to its current main definition, *the representation of an object, situation, or set of information as a chart or other image* [11]. It is clear through this example that visualization is evolving, and so is its role in society. We acquire more information through vision than through all of the other senses combined [12], and in a world where we gather more data than ever before, there is an undeniable motivation to use and improve the current data visualization techniques. One of the greatest benefits of data visualization is the great quantity of information that can be rapidly interpreted if it is properly presented.

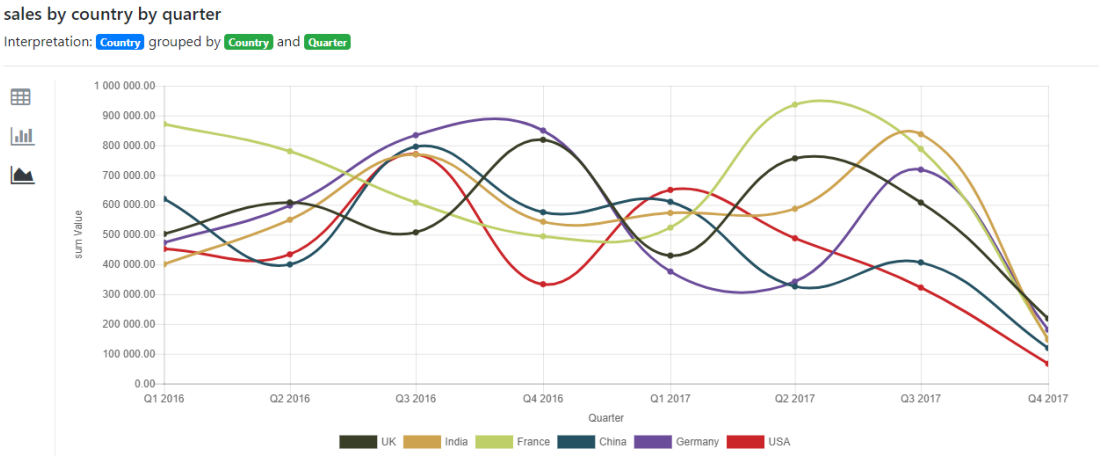


Figure 3.1 – Visualization Example from Ansa

A good visualization, as described by E. Tufte on the topic of *graphical excellence* [13, p. 48], should give the viewer *"the greatest number of ideas in the shortest time with the least ink in the smallest space"*. These ideas should be communicated with clarity, precision and efficiency, while truthfully representing the data.

Some of the advantages of a good visualization are [12]:

- Providing the ability to comprehend great amounts of data;
- Providing the perception of emergent properties that were not anticipated. The perception of a meaningful pattern can often be the basis of a new insight;
- Enabling problems with the data itself to become apparent. A visualization commonly reveals information not only about the data itself, but about the way it is collected. With an appropriate visualization, errors and artefacts in the data often stand out, making visualizations a valuable tool for quality control.
- Facilitating the understanding of large-scale and small-scale features of the data. It can be especially valuable in providing the perception of patterns linking local features.

While there is not a master formula for creating a visual representation, there are some relevant points to be regarded. Ware [12] has presented the process of data visualization as four interconnected stages: The collection of data and its storage; the pre-processing designed to transform the data into understandable pieces; the data processing and display, through algorithms that turn the information into a visual representation; and the perceiver of the visualization, presumably the human perceptual and cognitive system. His assumption is that once the best way to visualize data for a particular task is defined, it is possible to construct algorithms to create the appropriate visualizations.

In his literature, R. Mazza [6] divided the procedure to create a visual representation in five steps:

- **Defining the problem**: It is recommended that time is spent with the potential users of the representation, to identify their effective needs and how they operate. This is necessary to define what needs to be represented, how it should be represented, and, also important, why it needs to be represented. The visualization may be needed to communicate with a particular target, for finding new information, or possibly to prove a hypothesis. It is necessary to bear in mind the human factors specific to the target audience of the application and their cognitive and perceptive abilities. This will influence the choice of which visual models to use, to allow users to better understand the information.
- **Examining the nature of the data to represent**: It is possible to divide data in three main categories. It can be quantitative (e.g., a list of integers or real numbers), ordinal (of a non-numeric nature, but which has its own intrinsic order, such as the days of the week), or categorical (data that has no intrinsic order, such as the names of people or cities). Each of these types of data may require a different kind of processing and mapping.
- **Determining the number of attributes of the data**: The kind of visual representation used is strongly related to the number of attributes (also called dimensions) of the data. Attributes can be dependent or independent. Dependent attributes are the ones that vary, and therefore the ones for which there is an interest in analysing (e.g., when examining a product's sales over time, the sales would be a dependent attribute, while the time measure would be independent).
- **Creating data structures**: These can be linear (data is codified in linear structures such as vectors, tables, etc.), temporal (data that changes in time), spatial or geographical (data with a physical correspondence), hierarchical (data relative to entities organized on hierarchy), and network structures (data that describes relationships between entities).
- **Determining the type of interaction**: As a final step, the desired type of interaction should be determined. Depending on the purposes of the visualization, it may be static (e.g., an image printed on paper or represented on a computer screen but not modifiable by the user), transformable (when the user can control the process of modification and transformation of data, such as varying parameters of data entry, varying the extremes of the values of some attributes, or choosing a different mapping for view creation), or manipulable (the user can control and modify parameters that regulate the generation of the views, like zooming on a detail or rotating an image represented in 3D).

3.1.1 Concepts

It is important to establish a solid understanding of some fundamental concepts on the topic of visualization, as they will be the basis for following the challenges and proposed solutions of this thesis.

Chart

In the context of data visualization, a chart is a graphical representation of data, in which the data is represented by symbols, such as bars in a bar chart, lines in a line chart, or slices in a pie chart [14]. There are several kinds of chart and while each of them has its own characteristics, they share the same purpose of achieving the goals mentioned in the previous section. As C. Woods [15, p. 26] put it, *"charts can express more on one page than is sometimes expressed in several chapters of writing"*.

The selection of which chart type to use depends on the data that needs to be displayed. For example, data that presents percentages in different groups (such as "satisfied", "not satisfied" and "unsure") is often displayed in a pie chart, but may be more easily and accurately understood when presented in a bar chart [16]. On the other hand, data that represents numbers that change over time (e.g., a company's sales from 2015 to 2017) might be best shown as a line chart. In some cases, the reason for retrieving the data itself may influence the best chart type to choose, or the specific chart parameters to use.

Generally, the most relevant elements that make up a chart are:

- **Item**: An item, also called data point, is a single element in a chart (e.g., a single bar in a bar chart, a single point in a scatter plot, a slice from a pie chart).
- **Axes**: References for visualizing the data. The amount of axes necessary for a chart depends on the dimensions of the data being represented. Each axis will normally have a scale, denoted by periodic graduations and accompanied by numerical or categorical indications.
- **Values**: The set of measurements for each axis of an item (in a simple chart with two axes, these could be the typical X and Y values).
- **Series**: A collection of measurements. Generally, a set of data is called a series. The line connecting the data points in a line chart, for instance, represents a data series. If the same line chart has several lines, that would mean there are several series being displayed, using the same axes as reference.
- **Labels**: An important part of any chart which simplifies the process of understanding what is being displayed. Typically, there is a title along with the chart and each axis will also have a label displayed outside or beside it, briefly describing the dimension represented.

Chart Types

Chart configuration is one of the main areas of study of this project. In this section an overview of the most common chart types and their characteristics is presented.

Bar Chart

The bar chart (Figure 3.2) presents data with rectangular bars with lengths proportional to the values they represent. One axis of the chart shows the specific categories being compared while the other axis represents the corresponding values. Variations of this chart may use the bars horizontally or vertically, and the different series may be grouped in clusters side by side or stacked on top of each other, as shown in Figure 3.3, allowing for different kinds of analysis of the same data.

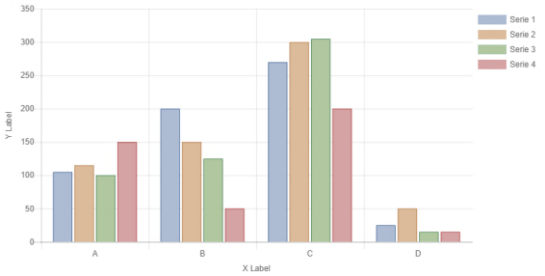


Figure 3.2 - Bar Chart Example

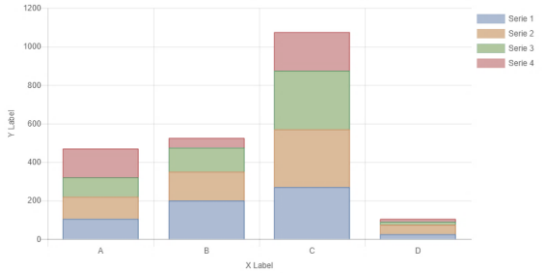


Figure 3.3 - Stacked Bar Chart Example

Line

Line charts (Figure 3.4) are similar to bar charts, but the data being displayed is continuous. The data points are connected either by a straight or a smoothed line to represent the data variation. It is also possible to stack the lines, as was done with the bar chart bars in Figure 3.3.

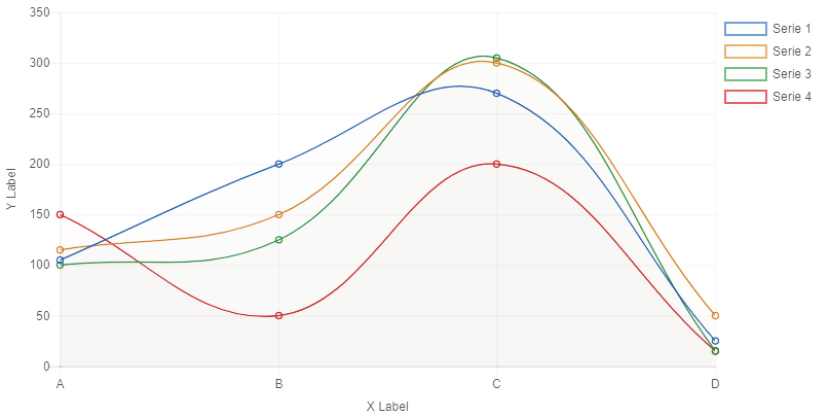


Figure 3.4 - Line Chart Example

Pie

Differently from the previous chart types, the pie chart (Figure 3.5) only has one axis. It is a circular graphic which is divided into slices that illustrate the numerical proportion of the data. It is widely used but also criticized for being difficult to compare different sections of a given pie chart, or to compare data across different pie charts [13]. Often, there may be some extra indications to help with these issues, such as displaying the actual value or corresponding percentage in each slice.

Common variations of the pie chart include the doughnut chart, which has a blank centre, allowing for additional information to be displayed, the exploded pie chart, where one or more sectors are highlighted by being separated from the rest of the circle, and the polar area diagram, where sectors have equal angles and differ rather in how far each one extends from the centre of the circle.

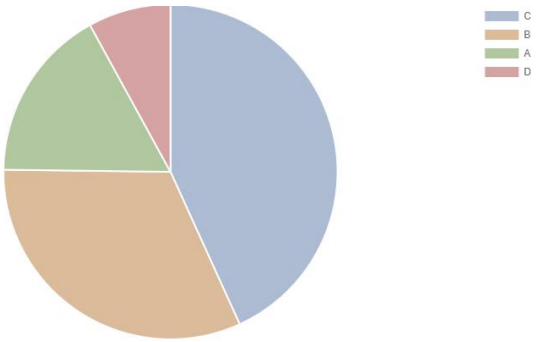


Figure 3.5 - Pie Chart Example

Scatter Plot

The scatter plot (Figure 3.6) consists in the display of data points in a grid and is useful to reveal relationships between the variables being analysed.

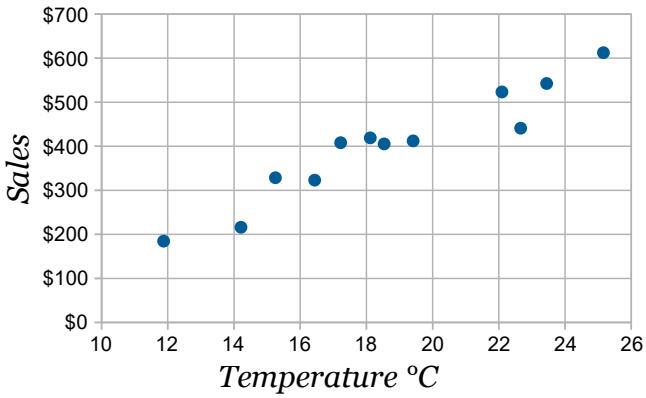


Figure 3.6 - Scatter Plot Example

Histogram

A histogram (Figure 3.7) is a plot that demonstrates the underlying frequency distribution of a set of continuous data. This enables the inspection of the data for its distribution (e.g., normal distribution), outliers, skewness, etc. [17]. In this chart, the data is arranged into intervals, called bins, dividing the range of values in equal parts and displaying the frequency of the data in each of the bins. Unlike bar charts, histograms are only used to plot the frequency of occurrences in continuous data, therefore there are no gaps between the bars (although some bars might be absent, indicating no frequency of occurrence). In contrast, bar charts may be used for a great number of other types of variables, including ordinal and nominal data sets, which makes them more adaptable.

Radar

Radar charts (Figure 3.8), also known as spider charts, consist on the representation of three or more variables represented on axes starting from the same point. The relative position and angle of each axis is typically uninformative. It is a useful technique for analysing which datasets are most similar and to find outliers in the data.

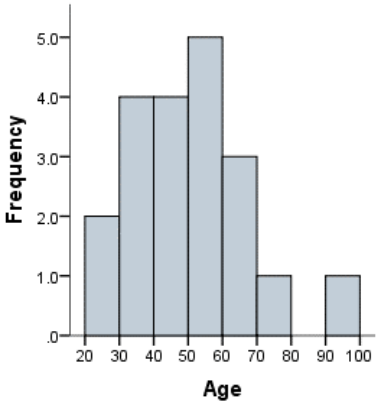


Figure 3.7 - Histogram Example

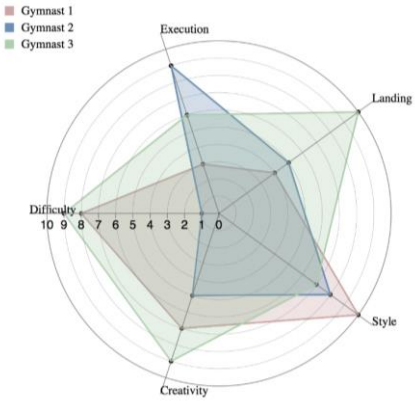


Figure 3.8 - Radar Chart Example

3.1.2 Evaluation

At the start of this chapter we discussed what constitutes good visualizations. This, however, might not be enough to know how to evaluate them. Choosing the appropriate chart type for a given data is not a trivial task. Not only there is not a particular set of rules, but also it seems that the *proper* chart type to use may depend on who will be using the data. Two users faced with the same visual representation might express completely different and contrasting judgments. Experience, prior knowledge, and perceptive and cognitive ability differ from person to person, which can result in diverging opinions [6]. For example, some might say pie charts are bad for visualizing data. Tufte goes as far as labelling them as dumb in *The Visual Display of Quantitative Information* [13], based on it leaving the viewer with the uneasy and uncertain task of estimating and comparing areas or angles in order to take conclusions. According to him, even a simple table will almost always be a better option. Yet, it is an undeniably popular chart type that can often be found in business and media.

As such, it is necessary to define some metrics to assist in the task of evaluating the visualization system. According to Mazza [6], the main objectives of study in the evaluation process are:

- **Functionality**: Does the visual representation provide all of the functionalities requested by the user and identified during the requirements definition stage?
- **Effectiveness**: Does the visual representation provide the users with a better knowledge of the data than the traditional non-visual methods would provide? In particular, does the use of visual representations allow the users to have more useful and precise information on the data than other tools? Or, is there additional information that is made available exclusively by the visual representations?

- **Efficiency**: Can the visual representation provide the users with information more rapidly than other tools?
- **Usability**: Is the interaction with the graphical interface simple and intuitive enough for the user?
- **Usefulness**: In what way, and in what context, is the information provided by the graphical representation useful to the user?

Depending on the type of application and the resources available for the evaluation process, it is possible to only evaluate a subset of these parameters. For example, in some cases it might be enough to evaluate the functionality, usability and effectiveness of a representation, assuming the usefulness is intrinsic in the desire to implement this feature and that efficiency is not a critical factor for success or, possibly, it is not yet a concern at a given development stage.

Since methodologies specific for evaluating systems of visual representations do not exist, techniques from human computer interaction have been adopted [6], which can essentially be subdivided into analytic methods and empirical methods.

Analytic evaluation methods are mainly based on cognitive and behavioural studies. They are carried out by experts who use the system to verify if it is compliant with a series of predetermined heuristics. An evaluator may also define a series of use cases and simulate the user's behaviour attempting to perform predetermined tasks, identifying possible problems that could originate from each of them.

Empirical methods make use of experiments using working prototypes of the system and its final users. The experiments may be quantitative, using controlled experiments, or qualitative, using interviews with the users, taking note of direct observations, creating focus groups, etc.

3.2 Automatic Visualization Projects and Techniques

Generating visual representations from user's data is an ambition shared across different projects. Some examples backed by powerful companies are IBM's Watson Analytics[1] and Microsoft's Power BI[2], both of which include the possibility of interacting with the system using natural language. Just as Ansa, they intend to be used across different sectors and enable users without data analysis expertise to examine their data, discover relationships, test correlations and look for any kind of insight of their interest. These projects, however, do not go into detail on their implementation of automatic data visualization.

An early approach to this challenge, carried out by Jock Mackinlay [18], attempted the development of an application-independent presentation tool that would automatically design effective graphical presentations of relational information. According to the research, the main problems faced were the encoding of graphic design criteria in a way that could be used by the tool and the generation of a wide selection of designs to represent various types of data. Once provided with the data for representation, the system

[1] https://www.ibm.com/watson-analytics
[2] https://powerbi.microsoft.com/

used a divide-and-conquer style algorithm, with three steps: partition, selection and composition. Each step involved choices to create composite designs. When this was not the case, backtracking was used to generate different solutions. The logic program developed included about 200 logic programming rules, making it a relatively complex solution.

Show Me [2], a project integrated in a commercial visual analysis system called Tableau[3], relies on the user specifying which data or data transformation (e.g., the values sum) should be used as columns and rows, and uses an internal ranking for chart types based on their popularity and difficulty of interpretation. For its response, the data is analysed and a set of rules determines which chart types can be used, from which the best ranked one is selected. The rules are mainly based on the data type and number of attributes. There is also a set of chart types that is not ranked and not proposed by the system, but may be selected by the user if more visualization alternatives are requested. In total this system includes 14 types of visualization, some of which are very specialized for particular types of data, meaning they are highly ranked yet rarely used as the conditions for their application are rarely satisfied (an example was the Gantt chart).

Articulate [19] intended to tackle the same challenge and also included NLP processes. In this system, the user's query is examined to identify what kind of analysis is being requested (relationship, comparison, composition or distribution). From then on, a set of rules related mainly to the type and number of variables determines which of the visualization options should be used, similar to what is done in Ansa.

DataTone [20] goes further than Articulate, enabling the users to follow up on their query, adjusting the output visualization accordingly. From the user query and its output, DataTone creates what it calls *visual specifications* to generate the appropriate visualizations. The system creates more than one visualization for the problem and provides the one it considers the best, while retaining information on the ambiguities encountered in the process. To deal with such uncertainties, this response is presented along with widgets which allow the user to intervene and adjust it as they see fit, correcting the decisions made by the system.

The *Natural Language Driven Data Visualization* project (NL4DV) intended to create a toolkit to provide developers and designers of visualization systems with high-level functions for using natural language query interfaces as extensions to their existing systems or build visualization systems entirely driven by natural language [21]. Its main modules consisted of a query processor, an attribute extractor, a task identifier and a visualization recommendation engine. The attribute extractor uses information from the query processor to identify the data attributes mentioned in the user's input, while the task identifier generates a list of tasks implicitly or explicitly stated in the input query. It recognizes 10 types of task (retrieve value, filter, compute derived value, find extremum, sort, determine range, characterize distribution, find anomalies, cluster, and correlate). Provided with the recognized attributes and tasks, the visualization recommender module selects the chart type and how to use the attributes. While the method used is not explicitly specified, it is also presumably a rule-based system.

Several approaches in this field of study are focused in the challenges related to processing natural language and fetching and preparing the most relevant data, rather than

[3] https://www.tableau.com

in the generation of the visual representation itself, which tends to be a variation of a RBS with different degrees of complexity. However, some experiments with case-based reasoning can also be found in literature.

The work of Michelle Zhou and Min Chen [22], from the IBM T. J. Watson Research Centre, was the first to attempt this approach. Motivated by the skill-dependent and time-consuming task of creating effective visual presentations, they developed a case-based visualization generation algorithm, which uses a database of existing graphic examples to automatically create visual presentations for new user requests. This work achieved good results, presenting more versatile design suggestions than a standard rule-based approach does according to their tests based on user feedback. The documentation puts an emphasis on three unique features: the adequacy-guided case retrieval method, which consists of a similarity metric with a set of adequacy evaluation criteria to retrieve the top-matched usable case; the case-base organization, which enhances case retrieval speeds by organizing cases into hierarchical clusters based on their similarity distances and by using dynamically selected cluster representatives; and an adaptation technique, which creates a solution based on multiple cases.

In "Creating Visualizations: A Case-Based Reasoning Perspective" [23], Freyne and Smyth attempted a simpler approach. They intended to assist users of a web based visualization system in producing better visualizations by recommending visualizations that had been previously used for datasets similar to their own. Their approach was limited: there was a reduced number of case features and the system's response would only suggest the best chart type to use, not indicating how to organize the data attributes into axes and series, for example. Yet, the system performed very well in practice, outperforming the remaining techniques in their tests, which were based on random selections, popularity, or exact matching of the input.

3.3 Case-Based Reasoning

Case-based reasoning is a problem-solving technique. Its most defining feature is that rather than relying uniquely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions to find the solution for a problem, CBR is able to use the specific knowledge of previously experienced, concrete problem situations, referred to as *cases*. A new problem is solved by finding a similar past case and reusing it in the new problem's situation. A second important feature is that CBR is an approach to incremental, sustained learning, since each new experience is retained every time a problem has been solved, making it immediately available for future problems [24]. As the system solves new problems, its case-base (the collection of known problems with verified solutions) grows, resulting in better proposed solutions for the problems to come.

It is common to find this kind of reasoning in day-to-day situations [25]. Ordering a meal at a restaurant, our choice will often be based on previous experiences in that restaurant or similar ones. While arranging our weekly schedule, we rely on the past to determine how long certain tasks take, and use that knowledge to adjust our plans. Likewise, a mother breaking up a fight between two children remembers what worked and didn't work previously to calm such situations, and bases her actions and suggestions on those experiences.

Examples of CBR-like decision making are many and are not contained within common sense situations. A medical doctor examining a patient with distinct symptoms might determine the most appropriate treatment by recalling another patient that was treated weeks earlier and showed similar, relevant, key characteristics. Properly evaluating the relevancy of case characteristics is an important part of the process. Possibly, a previous patient was also wearing a white shirt and jeans, but this would (almost certainly) not be related to the illness, so a higher quality comparison can be made when focusing on the previous patients' symptoms instead. Likewise, a drilling engineer, who has experienced dramatic blowout situations, is quickly re-minded of these situations when the combination of critical measurements matches those of a blowout case. He might also be reminded of a mistake he made when the blowouts occurred and use this to avoid repeating the error. A financial consultant working on a difficult credit decision task will be influenced by his experience with previous cases where companies with similar problems as the current one failed, and recommend that the loan application should be refused.

The examples are numerous and, software-wise, this technique has shown the potential for great results when many well-documented histories of past problems and their solutions exist [4][26]. In the following sections the main CBR concepts and stages will be described.

3.3.1 Case

In advance of the following sections, it is important to have a clear idea of what a case is.

A case is a contextualized piece of knowledge representing an experience. In general, a case may consist of a problem description, which represents the state of the world when the case occurred, a problem solution, which represents the derived solution to that problem, and/or an outcome, which describes the state of the world after the case occurred [27].

Cases that comprise problems and their solutions can be used to derive solutions to new problems, while cases comprising problems and outcomes can be used to evaluate new situations. If such cases also contain solutions, they can be used to evaluate the outcome of proposed solutions and prevent potential problems. Cases with higher amounts of information can be more useful, but they increase the system's complexity, making it harder to use. For this reason, most CBR systems are limited to storing only problem descriptions and solutions.

Cases can be represented as simple feature vectors, or using any typical AI formalism such as frames, objects, predicates, semantic nets or rules. [27]. The choice of which representation formalism to use is largely determined by the information to be stored within a case.

Throughout this document two main types of case will be mentioned: cases from the case-base, which are problem-solution pair; and *problem cases*, which represent the new problem being input to the system and do not have a defined solution. Later on, in the tests and results section (Chapter 6), *test cases* will also be mentioned. These are similar to case-base problem-solution cases.

Cases are the basis of any CBR system: a system without them would not be a case- based system. However, a system including only cases and no other explicit knowledge (not

even in the similarity measures) is difficult to distinguish from a nearest-neighbour classifier or a database retrieval system [27]. In other words, such system does not exploit the generalisation and adaptation potential of CBR, usually resulting in poor performance due to inefficient retrieval based upon case-by-case search of the whole case-base.

3.3.2 CBR Cycle

Generally, a typical Case-based Reasoning cycle may be described by the following four processes, sometimes referred to as *the four REs* [24]:

1. **Retrieve** the most similar case or cases;
2. **Reuse** the information and knowledge in that case to solve the problem;
3. **Revise** the proposed solution;
4. **Retain** the parts of the experience likely to be useful for future problem solving.

A new problem is solved by retrieving one or more previously solved cases, reusing the case in some way, revising the adapted solution and retaining the new experience by incorporating it into the existing knowledge-base (case-base). This cycle is visually represented in Figure 3.9.

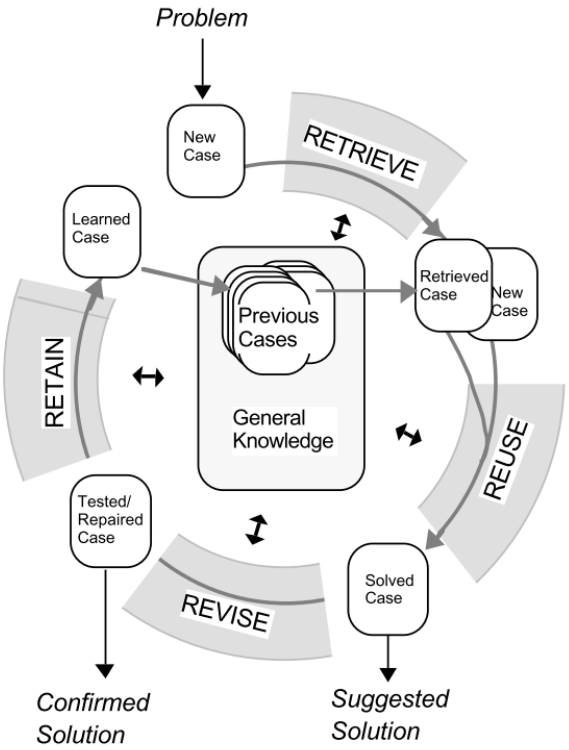


Figure 3.9 - Case-based Reasoning Cycle [24]

In detail, a new case is generated from an initial description of the problem. The difference between the problem itself and the case generated from it is that the case only includes some of the problem's information. The data that is considered irrelevant is removed from the problem, keeping only the features of interest that are intended to be analysed by the system and compared to other known cases. This new case is used to

retrieve a case from the case-base, the one that is found to be the most relevant to the new problem. The retrieved case is then combined with the new problem case, and is reused to propose a new solution, solving the case. This proposed solution goes through a revision phase, where it is tested for success, either by being applied in a real-world environment or evaluated by a teacher. According to its performance or feedback it may or may not need to be repaired. During the retention process, the experience is retained for future reuse, and the case-base is updated by the introduction of a new learned case, or by modification of some existing cases.

Based on this cycle, it understandable that the quality of a case-based reasoner's solutions depends on five main aspects [25]: The experiences in its knowledge base, its ability to understand new situations based on those old experiences, its aptitude at adaptation, its aptitude at evaluation, and its ability to appropriately integrate new experiences into its memory.

3.3.3 Case-Base Organization

Case storage is a vital aspect when it comes to designing an efficient CBR system, as it is responsible for providing the means to create an efficient and accurate search. These two qualities are usually inversely proportional [27], as an accurate retrieval guarantees that the best matching case will be retrieved, and efficient retrieval guarantees that cases will be retrieved fast enough for acceptable system response times. It is easy to ensure accurate retrieval if efficiency is sacrificed (e.g., by matching all the cases), as it is simple to have a fast retrieval if only a fraction of the employed case-base is searched, possibly missing relevant examples. Therefore, the challenge is to tune the case-base organization and the retrieval algorithm in order to adjust this compromise between accuracy and efficiency of the retrieval algorithm.

In general, there are three main approaches for case-base organization: flat, clustered and hierarchical organization.

Flat organization is the most straightforward case-base type, where the case-base follows a flat structure, such as a list. It is simple to create and to maintain, and retrieval usually searches case-by-case through the whole list. This means it can be very accurate, but also less efficient, especially for medium and large case-bases.

The clustered organization approach is based on gathering clusters of cases. Clusters may be based on mutual similarity or on the similarity to some prototypical cases. The advantage being that the selection of the clusters to be matched would be simple as it would be based on the indexes and/or prototypical cases characterising the clusters. A disadvantage is that this makes adding or deleting a case more complex than in a simple flat organised case-base.

A hierarchical organization is the case-base organisation that is generally obtained when cases that share the same features are grouped together. The case memory is organized as network structure of categories, semantic relations, cases, and index pointers. Each case is associated with a category and the categories are inter-linked within a semantic network. A common approach is having abstract cases, offering high level solutions, and concrete design cases, making it possible to decompose target problems into simpler sub-problems, reusing parts of complex problems as individual cases and recombining solution parts into a coherent whole [28].

3.3.4 Similarity Measure

A retrieval algorithm should retrieve the cases that are the most similar to the current problem being analysed by the CBR system. This task of comparing cases in order to determine the degree of similarity between them has been executed with a variety of methods, such as induction search, nearest neighbour search, serial search, hierarchical search, parallel search, etc. [27]. It is considered to be one of the core and most challenging tasks in developing a good CBR system [29].

The simplest form of retrieval is the 1st-nearest-neighbour search of the case-base, which performs similarity matching on all the cases in the case-base and returns the best match. As one might expect this method may result in long retrieval times, especially for large case-bases. Therefore, cases are usually preselected prior to similarity matching. Cases can be preselected using a simpler similarity measure that excludes the ones that are not considered a match and as such are not worth the time for performing the full similarity measure process.

3.3.5 Adaptation

Once the best matching case is selected for use, it may not be directly usable to solve the problem or situation being processed. Adaptation looks for prominent differences between the retrieved case and the current case, and, generally, applies a formula or a set of rules to account for those differences when suggesting a solution. It is a domain specific task, for which there are two general approaches [30]:

- **Structural adaptation**: applies adaptation rules directly to the solution stored in cases. If the solution comprises a single value or a collection of independent values, structural adaptation may involve modifying certain parameters in the appropriate direction, interpolating between several retrieved cases, voting, etc. However, if there are interdependencies between the components of the solution, structural adaptation requires a thorough comprehension and a well-defined model of the problem domain.
- **Derivational adaptation**: reuses algorithms, methods, or rules that generated the original solution to produce a new solution to the problem currently presented to the system. As such, this approach requires knowledge about how the solution was achieved in the first place, so it is very domain specific and can only be properly performed when there is an underlying understanding of how the recorded solutions were selected.

3.3.6 Advantages

As a reasoner, Case-based Reasoning provides several advantages [25].

To start with, it allows the reasoner to quickly propose solutions to problems without needing to derive them entirely from scratch. While it is always necessary to analyse the new case, with CBR there is a considerable amount of computations and inferences that are skipped. This advantage is helpful for almost all reasoning tasks, including problem solving, planning, explanation, and diagnosis.

Case-based reasoning also allows a reasoner to propose solutions in domains that are not completely understood. One might say that some domains are impossible to completely understand, such as areas which are strongly connected to human behaviour as is the economy, while others might just not be completely understood yet, such as some areas of medicine. In the medical doctor's example, from earlier in Section 3.3, it is possible that the patient will be cured as a result of the doctors reasoning and inferences based on the previous cases, and even though the doctor does not completely understand every detail on how treatment *A* works for the unknown illness with symptoms *X*, *Y* and *Z*, he does know it works, and he can retrieve and reuse that knowledge.

Moreover, CBR makes it possible for a reasoner to generate solutions when no algorithmic method is available for evaluation. It is possible that in some cases there simply is not enough knowledge to derive a dedicated algorithm for problem solving. Evaluating solutions based on previous similar situations might be the only sensible option, leaving the reasoner to make its decision based on what worked in the past.

Cases are also useful for interpreting open-ended, poorly defined concepts. A frequent example of this is the way attorneys use past cases and judge decisions as arguments for determining the outcomes of new cases. The domain may be lacking in theory and the interpretation of known cases may be used as a compensation method. The performance of Protos, a program designed to learn categories of a specific domain and classify new cases by explaining their similarities to known exemplars [31], in classifying hearing disorders when little information is known shows that a case-based methodology for interpretation can be more accurate than a generalization-based method when classifications are poorly defined.

Remembering previous experiences may also be useful for warning of the potential for problems that have previously occurred, alerting a reasoner to take actions to avoid repeating similar mistakes. Remembered experiences may be successful or failure episodes, i.e., situations in which the solution was not successful (or *as* successful) as expected. Also having the knowledge derived from such failed past case, the reasoner may be warned to avoid taking that exact combination of factors for solving the current problem.

Finally, cases help a reasoner focusing its reasoning in the key parts of the problem by pointing out the critical features. Important information in previous situations will tend to be important in new ones. If a set of features was implicated in the failure of a previous case, the reasoner focuses on those features to avoid repeating this failure in the future. Likewise, if a set of features have resulted in a successful case, the reasoner knows to focus on those features to propose a solution to the upcoming problems.

3.3.7 Pitfalls

Just as it happens with people's own reasoning, there are a few assumptions about the world under which CBR operates [32]:

- **Regularity**: the assumption that the same actions under the same conditions will have the same or similar outcomes.
- **Typicality**: the assumptions that experiences represent typical phenomenon that tends to repeat itself.

- **Consistency**: the assumption that small changes in the situation result in small changes in the interpretation and in the solution.
- **Adaptability**: the assumption that situations can be adapted to each other, and that small differences between the situations can be compensated for.

Having these in mind, there are some drawbacks to this kind of reasoning. A case-based reasoner might tend to use old cases blindly, relying on previous experience without validating it in the new situation [25], and whether the assumption that similar problems have similar solutions really holds may depend on the problem domain [4]. It is also possible that the case-based reasoner allows its case-base to make him/her/it biased in solving new problems. This may happen if the knowledge base does not properly and equally represent the full range of the problem domain. Moreover, the similarity metric depends on the domain and greatly affects CBR systems. Sometimes there may not be a good similarity metric to use at all, resulting in poor comparisons and weak foundations for the system's solutions. Likewise, adaptation is highly domain dependent and in extreme cases may be equally or more complex than a rule-based system.

As such, when applying CBR in a problem domain it is supposed that it satisfies the assumptions above. In the following chapter, the proposed approach for implementing the case-based reasoning automatic visualization system will be described.

Chapter 4

Proposed Approach

In this chapter, details about the proposed approach for implementing a CBR system to decide the best visual representation answer for Ansa's user queries are presented. It starts with information on the development methodology used throughout this project. Secondly, a use case is specified, which exemplifies a typical usage situation of the system and how it should behave based on user interactions. After that, the project's functional, technological and performance requirements are specified. The succeeding architecture section briefly explains Ansa's general architecture, as well as its rule-based system for handling visualizations. Finally, the proposed approach's architecture is explained, as well as each of its composing modules.

4.1 Development Methodology

The development of this dissertation's work follows the methodology adopted by the Ansa team: Scrum, which is an iterative and incremental development methodology used in Agile Software Development [33]. Scrum is a management, enhancement and maintenance methodology for an existing system or production prototype, which implements a set of development practices based on the assumption that the systems development process is an unpredictable and complex process that can only be roughly described as an overall progression [34]. It recognizes that it is likely customers will change their minds about what they want or need and that there will be unpredictable challenges along the road, for which a predictive or planned approach is not suited. As such, it adopts an empirical approach, accepting that the problem may not be fully understood or defined from the start, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements and to adapt to evolving technologies and changes in market conditions.

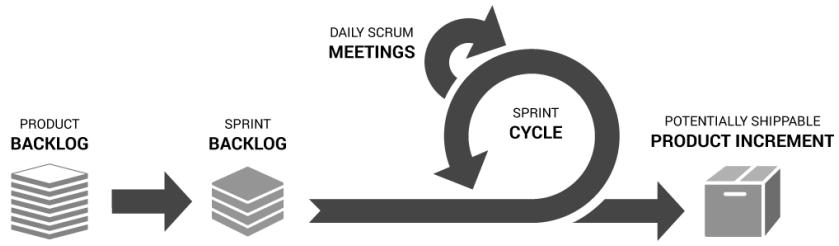


Figure 4.1 – General Scrum Workflow [35]

There are three core roles in the Scrum framework, which make the Scrum team. These are the product owner, the development team and the Scrum master. To summarize the process, the development team works as a unit to reach a common goal. Unlike the typical waterfall method, where the whole product is done in a sequential order of plan, build, test, review and deploy, having to go one step back whenever there is a problem, Scrum splits the work into several smaller pieces (Figure 4.1), the product backlog. There is a smaller planning phase, which is enough to get started on a reduced number of important features, the sprint backlog, which is then tested and reviewed. This process usually occurs within one to three weeks and is called Sprint. In Ansa's case, each spring consisted in two work weeks. By the end of each Sprint, the result is a potentially shippable product increment. The whole development is done sprint after sprint and throughout each Sprint, there is a short daily scrum meeting, where the team discusses what has been completed, what is being worked on, as well as any blockages that might have come up. At the end of each sprint, there is a sprint review and retrospective, where the team showcases their work to the product owner and works on what they can do to improve their processes.

The team collaborates using GitFlow, a Git branching methodology suited for collaboration and development team scaling [36], and manages the work associated with the scrum process using Jira[4], an issue tracking and project management tool, the most popular in its category [37].

4.2 Use Case

As mentioned above, the area of focus of this dissertation is the system responsible for deciding how to arrange the data retrieved form the database in an effective visual chart representation. This data is provided from the back-end as a *resultset* (see example in Appendix A), which consists on a selection of the relevant information resulting from the database query, arranged in table format. This selection is based on the interpretation of the user's natural language input. A high-level representation of this process is presented in Figure 4.2.

[4] https://www.atlassian.com/software/jira

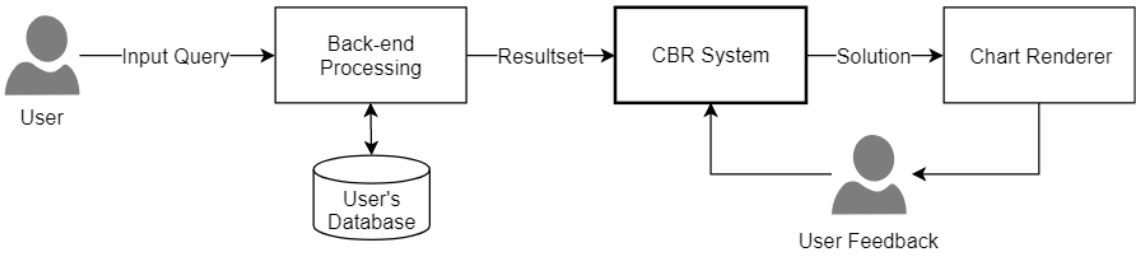


Figure 4.2 - Visual Representation Generation Flow

This system may have several users interacting with it, and they may or may not provide feedback on the system's response. If the feedback is negative a new solution is generated, while if it is positive the new case is added to the case-base. The following use case exemplifies different interactions between the user and the system:

- User #1 inputs a query which requires visual representation.
  → A visualization is generated with the most appropriate chart type and axes use.
- User #2 inputs a query which requires visual representation.
  → A visualization is generated with the most appropriate chart type and axes use.
- User #3 inputs a query which requires visual representation.
  → A visualization is generated with sub-optimal chart type and axes use.
  → User #3 is unsatisfied with the results and provides negative feedback.
  → The system generates a different visualization.
  → The user is satisfied with the results and provides positive feedback.
  → The system learns from the user feedback.
- User #4 inputs the same query as User #3.
  → A visualization is generated with the most appropriate chart type and axes use.
- User #5 inputs a query which requires visual representation.
  → A visualization is generated with the most appropriate chart type and axes use.
- User #6 inputs a query which requires visual representation.
  → A visualization is generated with the most appropriate chart type and axes use.
  → User #6 is pleased with the results and gives positive feedback to the system.
  → The system learns from the user feedback

4.3 Requirements

The definition of requirements plays a critical role in the success of software projects [38]. In this section a set of functional, technological and performance requirements are described. Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished [39], while non-functional requirements are related to the way those tasks should be accomplished.

4.3.1 Functional

The functional requirements of this project are described in Table 4.1.

Table 4.1 - Functional Requirements

ID	Name	Description
FR.01	Automatic visualization	Automatically decide appropriate visual representations for Ansa's search results based on confirmed known cases.
FR.02	Feedback system	Allow the user to provide feedback on the system's answer and adjust answer accordingly.
FR.03	Ability to learn	Expand its knowledge base as the system is used and faced with new problems and confirmed solutions.

These requirements indicate the main goals of the project. The validation of FR.01, the system's ability to decide appropriate visualizations, will be based on test cases with predetermined optimal solutions.

4.3.2 Technological

A requirement of the project was that it would be developed in the front-end, meaning the system would be running on the browser's side. This not only allowed to accelerate the development progress and focus on the case-based reasoning approach, as it served the purpose of experimenting with having the system moved from the back-end, a possibility which could provide improved performance and greater personalization options for Ansa. Since the system is meant to be running in the browser side, it was required for the programming language to be JavaScript.

The technological requirements of the system are described in Table 4.2.

Table 4.2 - Technological Requirements

ID	Name	Description
TR.01	Front-end development	The system must be developed in the front-end.
TR.02	Programming language	The system must be coded in JavaScript.

4.3.3 Performance

Considering Ansa intends to be used as an assistant, providing important information whenever necessary, it is expected that it has a near real time response, to ensure the users stay focused and their flow of thought remains uninterrupted. In the literature it is found that for this to happen the response should take a maximum of one second to be displayed [40][41].

This system, if integrated in Ansa, would only be a part of the full process from the moment the user inputs the query to the time the response is displayed, and thus the team defined that it could only require 500ms to be run.

The performance requirement of the system is described in Table 4.3.

Table 4.3 - Performance Requirements

ID	Name	Description
PR.01	Response time	The system's response time to determine the best visualization must be under 500ms on average.

4.4 Architecture

In this section the general Ansa architecture is presented, followed by a closer look at the current rule-based system in use for determining the visual representation to use, as well as the architecture of the proposed case-based reasoning approach to deal with the same challenge.

4.4.1 Ansa

Ansa is under development in a microservice architecture, which is based on structuring the application as a collection of loosely coupled services. Essentially, this means the software application is developed as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve the system's goal [42]. This kind of architecture provides modularity to the system and makes it easier to understand, develop, test and more resilient to architecture erosion. Another advantage is that it facilitates the team's methodology of continuous delivery and deployment [43].

4.4.2 Ansa's Visualization Rule-Based System

As mentioned in Chapter 2, Ansa relies on a rule-based system to select and configure the proper visualizations to use once the relevant data has been retrieved from the database. As demonstrated in Figure 4.3, this system starts by determining the possible charts to create with the received data. Once this step is complete, for each kind of possible visualization there is an extensive set of verifications and rules defining which attribute is used as the X-axis, if it is possible to have multiple series defined, and which attribute is used for the Y-axis, followed by a set of rules to create the chart configuration settings. In the end, another set of rules defines which is the most fitting chart to be displayed as Ansa's main response. After this process is complete its output will be used by the visualization rendering library to create the corresponding charts.

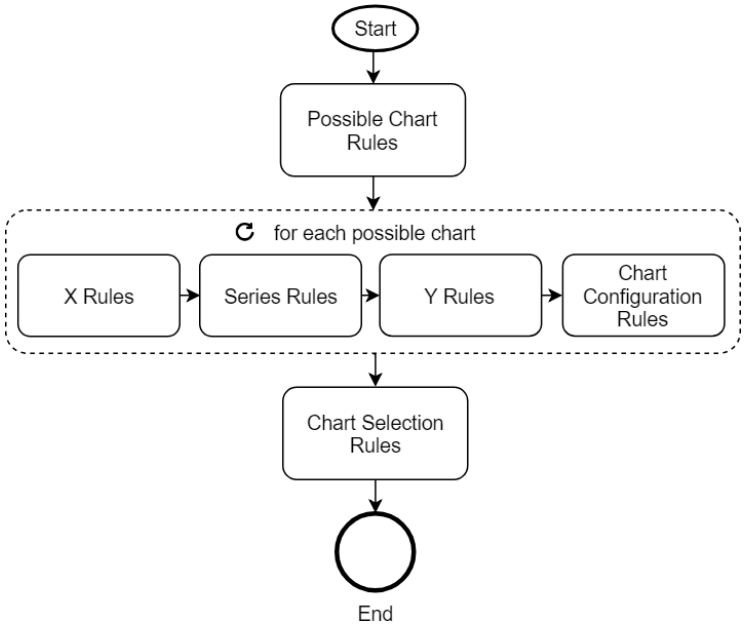


Figure 4.3 - Ansa's Visualization Rule-Based System

4.4.3 Proposed Architecture

It is possible to think of the proposed approach as having three main modules: the case-base ranking module, the solution generation module and the feedback management module. The way these modules are connected is represented in Figure 4.4.

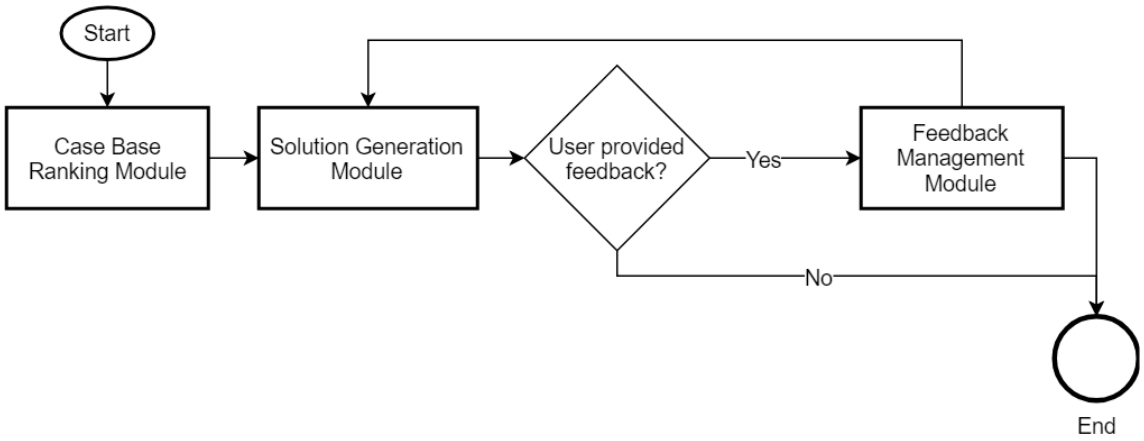


Figure 4.4 - Modules Overview

4.5 Modules

In this section each of the proposed architecture's modules is explained in detail.

4.5.1 Case-Base Ranking Module

Given a new problem, the case-base ranking module must provide a ranking list, containing the similarity measure of each of the case-base cases. When evaluating the

case similarity some cases are automatically considered unfit based on pre-selection criteria. These cases are skipped and do not get a rank value assigned. The ranking list will be used as one of the inputs for the solution generation module.

This module must be able to create a case by extracting the useful information from the input problem and saving it in a fixed format for processing. Afterwards it must have the ability to deserialize the case-base information and calculate the similarity between each of the cases and the problem case. These values will be kept in a ranking list of the same size as the number of cases in the case-base. This process is represented in Figure 4.5.

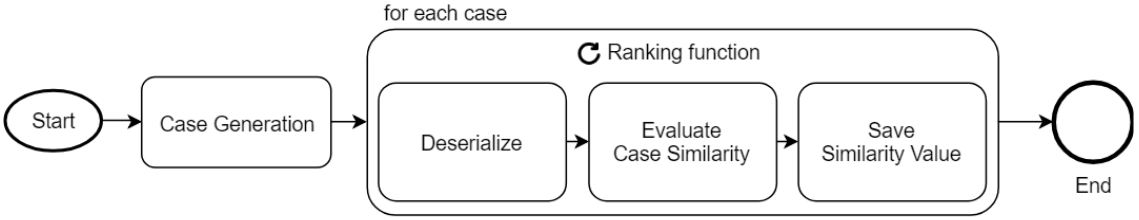


Figure 4.5 – Case-Base Ranking Module

4.5.2 Solution Generation Module

The solution generation module attempts to provide the user with the best possible solution based on the case-base knowledge. Alongside the new problem, this module also receives as input a list of ignored solutions, which is empty the first time the process is run for any given problem and is maintained by the feedback management module described in Section 4.5.3.

In this stage it is necessary to understand how to adapt the selected cases' solution to generate a valid solution candidate for the new case, and check if the adapted solution can be provided as a final answer or if it has been previously rejected by the user in the current use case. Subsequently, having found the best acceptable solution, it displays that solution to the user. This process is represented in Figure 4.6.

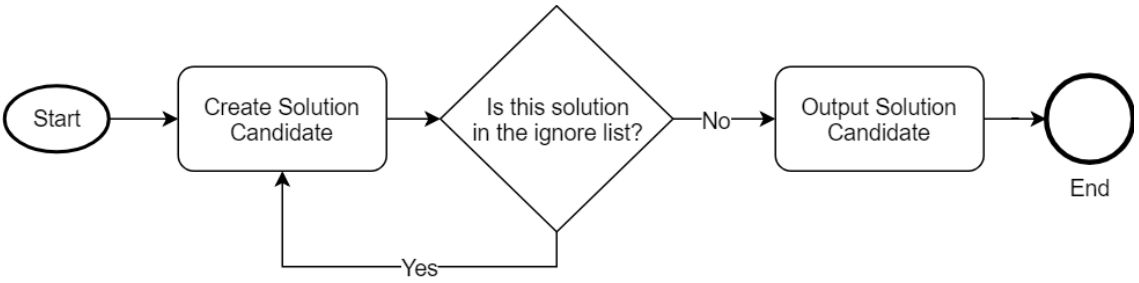


Figure 4.6 - Solution Generation Module

4.5.3 Feedback Management Module

Upon displaying a solution, the system allows the user to evaluate the results by providing positive or negative feedback. The feedback management module, as presented in Figure 4.7, initiates when this feedback is received.

The module starts by analysing the feedback. If it is positive, it is assumed that the generated solution constitutes the optimal visualization for the input problem, therefore the module stores the problem case along with the provided solution in the cases base, serializing it to match the case-base format. In the possibility of the problem case already having an exact match in the case-base, no storing action is performed.

In opposition, if the feedback is negative the provided solution is considered unfit and is added to a list of rejected solutions to be ignored the next time a solution is generated. Afterwards the solution generation module is restarted to provide the user with a different outcome.

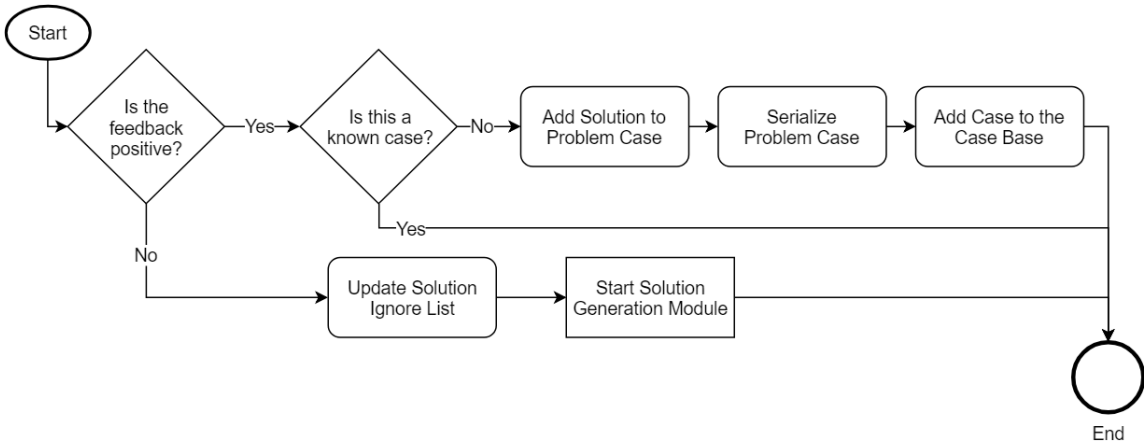


Figure 4.7 - Feedback Management Module

This chapter provided an overview of the fundamental processes behind the case-based reasoning approach. In the following chapter an in-depth view on its implementation details will be presented.

Chapter 5

Development

In this section a technical description of the development setting and implementation is presented, going through the development environment, resources and each of the CBR stages. Throughout the development and knowledge acquisition on the problem domain, new ideas for adjustments and improvements on the current implementation arise, resulting in promising thoughts for the following project iterations. These ideas are presented in Chapter 7.

5.1 Development Environment

In this section, an overview of the different platforms, tools and resources used throughout this work is presented.

5.1.1 JavaScript and Vue.js

The technological requirement TR.02 (listed in Table 4.2) imposes a JavaScript (JS) implementation. According to StackOverflow's[5] 2017 developer survey [44], JavaScript is the most popular programming language in use today. It is used by over 60% of developers and 95% of all websites [45], and is named as one of the top languages by popularity and demand by various sources [46][47][48][49]. JS is used on both client and server sides and helps to design intricate interfaces, enrich web apps with numerous functions and features and modify web pages in real time along with various other useful functionalities.

Often, JavaScript development is done using frameworks, which intend to provide several advantages. These are perceptible in terms of efficiency, with well-structured prebuilt patterns and functions, safety and even cost, as most frameworks are open source and free, helping programmers building custom solutions faster [50].

[5] StackOverflow (https://stackoverflow.com/) is the largest and most popular online community where developers learn and share programming knowledge.

These frameworks differ from JavaScript libraries in their control flow [51]: while a library offers functions that may be called by its parent code, a framework defines the entire application design [52]. This means a developer does not typically call a framework, instead it is the framework that is designed to call and use the code in a particular way (Figure 5.1).

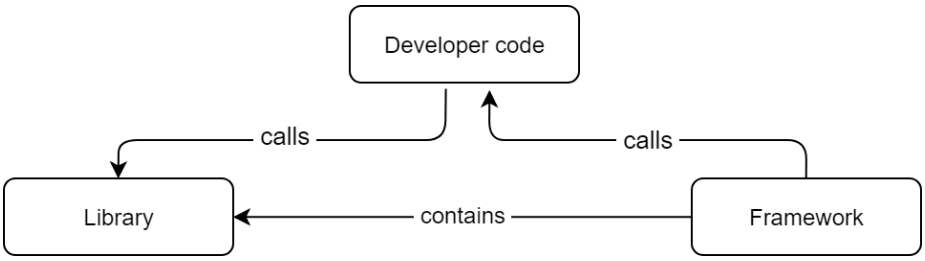


Figure 5.1 – Typical Framework and Library Control Flow

So far, the Ansa team has not been using a framework for their JavaScript codes, however, it is of their interest to experiment with this kind of tool and take advantage of its resources, which lead to the request for this CBR implementation to be executed using Vue.js[6], one of the top three JS frameworks.

Recently, the growth in support for Vue.js has been remarkable [53][54], greatly due to its small size, the simplicity to understand and develop applications with, good integration with existing applications, thorough documentation and flexibility [55]. Being third place in the list of most used frameworks, Vue.js follows AngularJS, developed by Google, and ReactJS, developed by Facebook. Its benefit is that it takes the best features from those frameworks and joins them in a convenient and accessible package. Comparing to its top alternatives, Vue.js is proved to be faster and leaner [50].

5.1.2 Tools and Resources

The work in this thesis was carried using a Bitbucket[7] repository, managed using its own client, Sourcetree. The IDE of choice for the development was WebStorm[8] by JetBrains, a powerful JavaScript IDE recommended by the team which provided all the necessary tools for the project.

Two libraries have also been used: Chart.js[9] and Axios[10]. Chart.js was used to create, animated charts by inputting the desired data, chart type and labels. Some visualizations generated with chart are demonstrated in Appendix E. It is the same library currently in use in Ansa for this purpose. Axios was used to consume and display data from Ansa's API. This promise-based HTTP client is a very popular approach for connecting to APIs when working under the Vue.js framework [56].

[6] https://vuejs.org/
[7] https://bitbucket.org/
[8] https://www.jetbrains.com/webstorm/
[9] http://www.chartjs.org/
[10] https://github.com/axios/axios

5.2 Case-Based Reasoning Solution

The implementation of a Case-Based Reasoning solution for determining the best data visualizations for Ansa's responses is the core goal of this thesis. The development of this solution consisted in an iterative approach, starting from a very basic design and implementation, which evolved throughout the development sprints, as new knowledge on the topic of CBR was acquired. As this work consists in a proof of concept for CBR application, its implementation was conducted externally, not in the actual Ansa system.

In this section, each of the main characteristics and processes that constitute the final solution are explained.

5.2.1 Case Representation

As represented in Listing 1, each case stored in the case-base contains general features, the attributes list and the solution. After analysing the problem, it was considered that these parameters, which were relatively easy to access or extrapolate from the problem data, could provide a solid base for case representation and comparison.

General Features

Part of the case specification, the *generalFeatures* contain general information about the case data for which the included solution is the optimal one. It includes as properties the number of columns, number of rows, number of columns with temporal information, number of columns with nominal information and number of columns with numeric information. It is meant to provide easy access to defining characteristics of the whole case without needing to further analyse the attributes.

Attributes List

Part of the case specification, *attributes* is an array of attribute objects. Thus, this is an array of variable size. Its elements, however, are of fixed length and each of them include the same properties. The first one, *id*, is present for identification purposes. The next three properties contain relevant information about the attribute, obtained directly from the resultset. The *transform* indicates if the attribute values have gone through a back-end transformation (e.g., timestamps may be transformed into nominals to allow the data to be grouped by month and presented with the corresponding labels). The *valueType* indicates the current type of values of the attribute, which may be nominal, numerical or timestamp. The *attributeName* is the name of the variable the attribute represents and *attributeType* is the type of attribute prior to its transformation (for instance, in the case of a date attribute from the query "Sales by month" in one of Ansa's test databases, the *attributeType* is nominal, since months will be represented as a word in the chart, while the *valueType* is timestamp, since that is how date information is stored in the database). Following these properties comes *max*, *min*, *median* and *stdev*. These are used for keeping numerical information if the attribute is numerical. Otherwise, they remain as zero[11].

[11] Technically, for a non-numeric attribute it is not correct to say that its maximum, minimum, median or standard deviation are zero. However, for the time being it was useful to store them to zero instead of *null*. Considering the solution never uses these values unless the attribute is numerical, this does not affect the results.

Solution

Representing the case solution, the *solution* object has two properties. The first one, *dataViewType*, indicates only the best chart type to use for the data, while the second one is an object that specifies the use for each of the case attributes. In the case of a bar or line chart, this means there will be an *axisMap*, indicating which attribute should be used as the Y-axis, which one should be used as the X-axis, and which one, if any, should be used as series. For pie charts, there is *pieChartMapper* which indicates which attribute should be used for the value, and which should be used for the labels.

When a new problem is given to the system, via a *resultset*, a case is generated from it. This case, called *problem case*, has the same structure as the case-base cases, except for the solution, which, naturally, is not yet specified as it is still undetermined.


```
{
    "generalFeatures": {
      "nCols": 2,
      "nRows": 10,
      "nDateCol": 1,
      "nNominalCol": 0,
      "nNumericCol": 1
    },
    "attributes": [
      {
        "id": 0,
        "transform": "MONTH",
        "valueType": "NOMINAL",
        "attributeName": "Date",
        "attributeType": "TIMESTAMP",
        "max": 0,
        "min": 0,
        "mean": 0,
        "median": 0,
        "stdev": 0
      },
      {
        "id": 1,
        "transform": "SUM",
        "valueType": "NUMERIC",
        "attributeName": "Value",
        "attributeType": "NUMERIC",
        "max": 2774810,
        "min": 1845907,
        "mean": 2350999.1,
        "median": 2406893.5,
        "stdev": 291234.84868691454
      }
    ],
    "solution": {
      "dataViewType": "PLOT_VIEW",
      "axisMap": { "Y_AXIS": 1, "X_AXIS": 0, "SERIES": null },
    }
  }
```


Listing 1 – A Case with Two Attributes

5.2.2 Case Retrieval

The current case-base implementation is using a flat organization, as explained in Section 3.3.3, meaning all the system's known cases are treated equally, stored as objects in the case-base array.

Even though cases are stored in JSON format for readability, when they are retrieved for use they are converted to a simplified representation which contains only the values of each of the properties arranged in array form. All the processing after their retrieval is conducted using this simplified representation. This distinct representation ensures that in the future it is possible to freely adapt the solution to experiment with other processing techniques. Machine learning techniques, for example, typically require the data to be in array format for processing. Meanwhile, keeping the case-base with the full JSON case representations ensures that it is easily readable by a human interactor, allowing for agile maintenance and examination if necessary. In Appendix C and Appendix D there is an example of each these case representation formats for comparison.

It is in the retrieval stage that the CBR process determines which case from the knowledge base should be used to create the solution for the problem case. This process consists mainly in running the known cases through a similarity function and selecting the most similar one.

Once the most similar case is found the reuse phase is initiated.

Similarity Measure

The similarity measure intends to calculate the distance between two cases. When the system is set to solve a new problem, it proceeds to rank all its known cases based on the results of this comparison. This is a crucial part of the CBR process [29] which has been implemented as follows:

1. The process begins with two inputs, the problem case, *P*, and the case-base case which is being analysed, *C*.
2. An initial selection is done with the goal of optimising the process. If the number of attributes of *C* is different from the number of attributes of *P*, *C* is considered unfit to generate a solution for *P*, and the measure is finished for this case, returning *NaN*[12].
3. The difference between the general features of each case is calculated. Each comparison starts from 0 and increases as the difference between each of the parameters is calculated. These comparisons consist in a weighted difference between each of the properties, meaning each case property has a pre-defined, adjustable weight multiplier, defined in a separate section. Their calculation is shown in Equation (5.1), where f_i^P is the value of feature i from case *P*, f_i^C is the value of feature i from case *C* and w_i is the defined weight for feature i.

$$numericDiff_i = \left| f_i^P - f_i^C \right| * w_i \tag{5.1}$$

   All the general feature values are numerical and are added up according to Equation (5.2), where *n* is the number of features.

[12] *NaN* is a numeric data type value representing an undefined or unpresentable value. It stands for *Not a Number*.

$$generalFeaturesDiff = \sum_{i=0}^{n} numericDiff_i \quad (5.2)$$

4. The difference between attributes is calculated. Each attribute of *P* should only be compared to its corresponding attribute in *C*, or whichever is the most similar to it (e.g., it would not make sense to compare attribute *Date* of *P* with attribute *Volume* in *C* if *C* also contains a *Date* attribute). Thus, before comparing attributes the system tries to find the closest possible match for each of them:

   4.1. One at a time, *C*'s attributes are compared to *P*'s attributes. Once the best match for a *n* attribute in *C* is found, the corresponding attribute in *P* is removed from the set and the process is repeated for the remaining attributes[13]. The attribute difference is the sum of the differences of each of the attributes' properties, which is calculated using Equation (5.1) for numeric properties, and using a method based on the Levenshtein distance [57], a popular similarity measure between two strings, for textual properties.

   4.2. Once the attribute matching is completed, each attribute in *C* is compared to its match in *P* and the differences are summed, as shown in Equation (5.3).

$$attributesDiff = \sum_{k=0}^{m} attributeDiff(k, k) \quad (5.3)$$

   where m is the number of attributes, k is the attribute being evaluated and k' is its matching attribute.

5. The attributes difference is summed to the general features difference, as shown in Equation (5.4). The resulting value is considered the total difference between *C* and *P*.

$$totalDiff = generalFeaturesDiff + attributesDiff \quad (5.4)$$

Using this algorithm for each case-base case, a case-base ranking is generated, from which the case with the lowest score (i.e., the lowest total difference) is considered the most similar to the problem case and is therefore used to generate a solution.

The ability to adjust the weights of each property provides a useful tool for testing and for understanding which are the most relevant factors when generating a new solution based on previous cases. These weights have a great influence the system's response, and require tuning, as expected.

5.2.3 Reuse

As previously demonstrated, the solution that the system generates for the problem consists in two elements. The first, and most straightforward one, is the chart type to be

[13] While this method prevents duplicate matches and increases system performance, it may not find the absolute best attribute matching combination.

select. The second one consists in information on how to use the available attributes to generate that chart.

```
"solution": {
      "dataViewType": "LINE_CHART_VIEW",
      "axisMap": {
        "Y_AXIS": 2,
        "X_AXIS": 1,
        "SERIES": 0
      },
    }
```

Listing 2 – Solution of a Case from the Case-Base

In order to reuse the retrieved case to generate a solution for the current problem, the system relies on two pieces of information: the retrieved case's solution and the attribute matching determined in the retrieval stage. This knowledge of the correspondence between the case and problem attributes is the basis of the developed CBR adaptation mechanism.

Using the retrieved cases' solution (example presented in Listing 2), we have direct access to the best chart type to use, and with it comes the information on how its attributes were used in that chart type. Following with the example from Listing 2, it is known that the solution consisted of a line chart using the case's attribute with id 2 as the Y-axis, id 1 as X-axis, and id 0 as the series. Combining this information with the attributes match defined when this case was compared to the problem case, it is possible to determine which of the problem case's attributes should then be used for each axis and series in order to replicate the same kind of successful result.

5.2.4 Revision

Once the solution has been generated, it is displayed to the user, which then has the opportunity to provide his/her feedback on whether it is optimal or not.

If the user is not satisfied with the provided solution, the system generates a new one. To do this, the previous solution object (chart type and axis mapping) is added to an ignore list, which includes all rejected solution candidates, and the CBR cycle is restarted. When selecting the new best case, cases with a solution combination (chart type and axis mapping) which, after undergoing the axis transformation described in Section 5.2.3, is already in the ignore list are skipped, ensuring the user receives as best solution that is not in the previously rejected solutions list. It would not be enough to simply exclude the cases that provided the rejected solutions from the case-base ranking, since different cases might be adapted to create the same solution.

Upon receiving the new solution, the user is prompted again for feedback, and the revision process can be repeated until the user is satisfied with a solution or until the system is unable to come up with new combinations to solve the problem.

Once the user is satisfied with the system's response, positive feedback initiates the case retention process.

5.2.5 Retention

Upon receiving positive feedback, the suggested solution is confirmed as the optimal solution and, along with the problem case it solved, is converted to the same readable JSON format that is used in the case-base (Listing 1) and added to it as a new case, increasing the system's knowledge and improving its future results, thus completing the case-based reasoning cycle.

Chapter 6

Tests and Results

The results of the performance and accuracy tests performed on the CBR system are presented in this chapter, followed by their analysis and discussion in Section 6.3. Table 6.1 describes the test machine specifications, common to all tests.

Table 6.1 - Test Environment

Specification	Test Machine
CPU	Intel® Core™i7-7700HQ @2.81GHz (4 cores, 8 threads)
RAM	16GB 2400MHz DDR4
Operating System	Windows 10 Pro
Solid-State Drive	Samsung PM981 SSD NVMe PCIe M.2 512GB

6.1 Performance

The following performance tests have been conducted using the performance interface from the High Resolution Time API[14], which allows the definition of key points in the code structure and the comparison of their occurrence time. The system was ran in Google Chrome, using version 66.0.3359.181 of the browser.

For the purpose of these tests, the response time will be calculated as the total time it takes from the moment the system receives the new problem to the moment it orders the solution to be displayed.

6.1.1 General Response Time

For this test, 39 resultsets were selected as representative of the various types of cases the system might be faced with, meaning the data was of all kinds, the number of attributes

[14] https://developer.mozilla.org/en-US/docs/Web/API/Performance

was not fixed and the solutions included various arrangements of the three types of visualizations currently available in Ansa: bar charts, line charts and pie charts. The test's result are presented in Table 6.2.

Table 6.2 - General Performance Test

Number of Test Cases	Number of Case-Base Cases	Average Response Time (ms)	Standard Deviation (ms)
39	12	4.05	1.20

6.1.2 Influence of Problem Case Attributes

In the following tests the influence in performance of the number of problem case attributes is inspected. For these values to be comparable it was necessary to ensure that both resultsets would be considered compatible to the same number of case-base cases, as incompatible cases are skipped, and compatible cases require the full similarity measure processing.

Each measurement was performed 2000 times. The average response time and standard deviation of the values is registered in Table 6.3.

Table 6.3 - Influence of Number of Problem Case Attributes in System Performance

Number of Attributes	Number of Compatible Cases	Average Response Time (ms)	Standard Deviation (ms)
2	3	3.18	0.85
3	3	3.73	1.25
2	5	3.48	1.35
3	5	4.09	1.15
2	50	16.91	6.40
3	50	17.13	4.94

6.1.3 Influence of the Case-Base Size

In the following tests the influence of the number of cases in the case-base in the overall system performance will be evaluated. Depending on the way cases are selected (which can be adjusted in future iterations), the number of cases in the case-base may not be, and ideally is not, directly proportional to the number of cases evaluated in each test. For this reason, some combinations of the total number of cases and number of compatible cases have been tested. The tests were completed using the same problem with 3 attributes and each measurement was performed 2000 times. The average response time and standard deviation of the values is registered in Table 6.4.

Table 6.4 - Influence of Number of Case-Base Cases in System Performance

Case-Base Cases	Compatible Cases	Average Response Time (ms)	Standard Deviation (ms)
5	5	2.65	0.97
10	5	3.63	1.23
15	10	5.16	1.91
20	15	6.27	1.86
25	5	6.10	2.20
25	10	6.53	2.52
25	20	7.91	2.69
55	5	10.76	4.65
55	35	15.89	4.45
105	85	31.56	9.92
205	85	63.06	16.51
505	485	156.93	53.14

6.2 Accuracy and Sensitivity

In this section we evaluate the system's accuracy and sensitivity, measuring the influence of each of the adjustable weights that take part in the similarity measure described in Section 5.2.2.

Initially, 51 *resultsets* from various kinds of query input in the different Ansa test databases were collected, from which 12 were randomly chosen[15], removed from the set and added to the case-base. The remaining 39 cases were used as the test set.

These tests were performed using a test routine implemented in the CBR system. When the routine is started it initiates a cycle which goes through the full test set and starts the CBR process for each of the test cases. By the end of each one it compares the generated solution (chart type and axes use) to the optimal solution, which is defined in the test case. After completing this process with every case, it indicates the percentage of solutions which used the correct chart type, the correct chart type and axes as well as a custom score metric explained below in this section.

The adjustable parameter weights in these tests are: number of columns, number of rows, number of columns with dates, number of nominal columns, number of numeric columns, attribute transformation, attribute value type, attribute name, attribute type, maximum value, minimum value, min value, mean value, median and standard deviation. These are the elements that define a case and its attributes, as is described in Section 5.2.1. The corresponding weights are stored in a simple array, as shown in Listing 3.

[15] Although the cases were selected randomly, there was one imposition for the selection to be accepted: at least one case of each chart type had to be included, ensuring the system possessed information on how to create the three types of answer.

```
export const caseBaseWeights = [
  1,    // nCols
  1,    // nRows
  1,    // nDateCol
  1,    // nNominalCol
  1,    // nNumericCol
  null, // attributes id
  1,    // attributes transform
  1,    // attributes valueType
  1,    // attributes attributeName
  1,    // attributes attributeType
  1,    // max
  1,    // min
  1,    // mean
  1,    // median
  1     // stdev
];
```

Listing 3 – Case-Base Weights Array

Naturally, not all of the parameters have the same importance, hence the implemented possibility to easily adjust their weight in the similarity formula. Since there is no rule to define which is more important or how they relate to each other, these tests will serve as a tuning tool and as a reflection on how the weights are currently implemented.

To verify which are the most influential parameters, a set of tests was conducted enabling each one independently. In this set of tests, the parameter being tested was set to 1, while the remaining ones were set to 0.

To have a better insight on the performance of the system in each test, a score metric was added. Instead of only considering if the system selected the perfect chart type and the perfect axis use, it defines that some sub-optimal outputs are better than others. This metric focuses on the chart choice and the axis use (which axes were chosen to represent which attribute), and defines that the chart type selection is of slightly greater importance than the axis attribution, weighting them as 60% and 40% for the final score, respectively. For each of these two parameters, tests may have different scores.

If the optimal visualization for a certain problem is a bar chart and the system fails to select it, choosing a line chart instead, the output is closer to the expected one than a pie chart would be. Despite the error of selecting a line chart for data that is not continuous, this response will still preserve the user's ability to comprehend and analyse the data to a greater extent than the alternative would. As such, it is considered that an answer that selects a bar chart instead of a line chart or vice versa has a 50% score on the chart choice parameter, *ChartType* in Formula (6.1). This value is only indicative that the answer is better than selecting a pie chart (which would be a 0% score on the *ChartType* parameter), and not as good as matching the expected chart type (which would result in a 100% score for this parameter). Table 6.5 indicates the different *ChartType* score combinations.

Table 6.5 - Possible *ChartType* Scores

Expected	Selected	Score (%)
Bar	Bar	100
	Line	50
	Pie	0
Line	Bar	50
	Line	100
	Pie	0
Pie	Bar or Line	0
	Pie	100

Likewise, when it comes to axes use, it is defined that failing to select the correct Y axis is a worse error than selecting the correct Y axis and switching the X axis with the series. This is because the X axis and series may be changed and still produce a comprehensible chart, while a different combination of the three likely results in senseless results. As such, as represented by Table 6.6, charts that only correctly used the Y axis have a 50% score in the *AxisUse* parameter. In the case of charts with only two attributes (as pie charts or some bar and line charts) the choice is either correct or incorrect.

Table 6.6 - Possible *AxesUse* Scores

Number of Axes	Correctly Used Axes	Score (%)
3	All	100
	Y	50
	Other	0
2	All	100
	None	0

The final score is calculated as follows:

$$Score = 0.6 * ChartType + 0.4 * AxesUse \quad (6.1)$$

where *ChartType* is the score achieved in the chart type selection parameter and *AxisUse* is the score achieved in the axis use parameter.

In each of the following tests, presented in Table 6.7, the influence of each parameter weight (as illustrated in Listing 3) is measured by deactivating the remaining ones. It is possible to analyse the percentage of correct chart type responses, the percentage of optimal solutions returned (when the solution was exactly the same as previously defined when collecting the test cases), the score metric and its standard deviation.

Table 6.7 - Individual Weight Tests

Test Number	Active Weight	Correct Chart Response (%)	Optimal Solution (%)	Score (%)	Standard Deviation (%)
1	(none)	50.00	18.42	49.21	30.21
2	*nCols*	50.00	18.42	49.21	30.21
3	*nRows*	52.26	15.79	44.74	28.88
4	*nDateCol*	89.47	31.58	66.84	29.12
5	*nNominalCol*	89.47	31.58	66.84	29.12
6	*nNumericCol*	50.00	18.42	49.21	30.21
7	*transform*	89.47	78.95	90.53	21.64
8	*valueType*	52.63	36.84	60.53	36.92
9	*attributeName*	89.47	84.21	91.58	21.59
10	*attributeType*	100.00	**89.47**	**97.89**	6.14
11	*max*	47.37	28.95	50.86	37.92
12	*min*	55.26	36.84	59.34	37.03
13	*mean*	47.37	31.58	50.72	39.20
14	*median*	50.00	36.84	55.26	39.02
15	*stdev*	57.89	39.47	60.66	37.12

After the 15 individual parameter tests, several weight combinations were defined to recognize how the different parameters are connected and which combination approaches the best possible results using the system in its current form. For these tests we will be manipulating the *caseBaseWeights* array, as represented in Listing 3. The tests' specifications were:

16. All parameters set to 1.
17. Test 16 with numeric parameters set to 0.
18. Only general features set to 1 (the first five elements of the weights array).
19. General features set to 1 and number of rows disabled.
20. Test 19 with attribute features set to 1.
21. Test 20 with attribute name disabled.
22. Only the value type and attribute type set to 1.
23. Only the attribute name and attribute type set to 1.
24. Attribute type set to 1 and the numeric features set to 0.0000001.

The registered results are presented in Table 6.8.

Table 6.8 - Weight Combination Tests

Test Number	*caseBaseWeights*	Correct Chart Response (%)	Optimal Solution (%)	Score (%)	Standard Deviation
16	[1,1,1,1,1,null,1,1,1,1,1,1,1,1,1]	47.37	34.21	51.32	39.77
17	[1,1,1,1,1,null,1,1,1,1,0,0,0,0,0]	86.84	81.58	88.55	26.68
18	[1,1,1,1,1,null,0,0,0,0,0,0,0,0,0]	78.95	23.68	57.57	29.74
19	[1,0,1,1,1,null,0,0,0,0,0,0,0,0,0]	89.47	31.58	64.01	29.83
20	[1,0,1,1,1,null,1,1,1,1,0,0,0,0,0]	100	**94.74**	**98.82**	5.02
21	[1,0,1,1,1,null,1,1,0,1,0,0,0,0,0]	100	89.47	97.63	6.91
22	[0,0,0,0,0,null,0,1,0,1,0,0,0,0,0]	100	89.47	97.63	6.91
23	[0,0,0,0,0,null,0,0,1,1,0,0,0,0,0]	100	**94.74**	**98.82**	5.02
24	[0,0,0,0,0,null,0,0,1,1,0.0000001 0.0000001,0.000001, 0.0000001,0.0000001]	100	**97.37**	**99.41**	3.60

6.3 Results Discussion

In this section a critical analysis of the performance and accuracy and sensibility tests is presented, followed by relevant final remarks.

6.3.1 Performance

The performance tests executed achieved very positive results. Even in the most demanding test, with over 500 case-base cases and almost the same amount of compatible cases, the average execution time was safely under the required limit of 500ms, demonstrating that for the existing kind of case a flat hierarchy solution is capable of dealing with great amounts of experiences and still remain with acceptable response times. Possibly, in the future, more parameters will be added to the case representation to improve the system's ability to distinguish cases, however it seems likely that this kind of adjustment would not change these results in a significant way.

It is also demonstrated that the influence in performance due to the number of attributes of the problem case is almost non-existent when comparing problems with 2 and 3 attributes. On the other hand, the number of cases, especially the number of compatible cases do cause perceptible, yet small, shifts in performance.

If this system is to be employed in Ansa, it would still have to be defined how learned cases are managed (some thoughts on this topic in Chapter 7), but for the moment being able to handle hundreds of stored cases in near real time is a satisfactory result.

6.3.2 Accuracy and Sensibility

The first set of tests (1 to 15) provide several insights on the influence of each weight.

Comparing test 1 to test 2, we realize that the weight of the number of columns is not currently affecting the system's accuracy. This happens because the similarity function already excludes from the comparison the cases which have a different number of columns.

Test 10 shows promising results which reveal that the most problem-defining property is the *attributeType*, an indication of what the attribute data represents (e.g., if an attribute contains the months of the year, even though each element is nominal, the *attributeType* is *timestamp*, not nominal). With this information alone, the CBR system is able to choose the correct chart type to represent the data in 100% of the test cases, and manages to create the optimal solution in 89.47% of them.

The attribute's name was also able to provide interesting results by itself, as shown in test 9. This property focuses on the actual name that is displayed with the attribute, its label. This means it depends on the dataset being used, and while it may not be particularly useful when working with general cases of mixed datasets, it can be good in differentiating cases when dealing with problems from the same dataset as the case-base, or when working with a small number of different datasets. In these tests, the test set and case-base cases were from three distinct datasets. In Ansa's case, the system would likely be provided to the customer with a standard case-base to start with. Using the name comparison for those cases might not make sense, as the customer's data could be completely unrelated, but the name property can be used later on to improve the comparison among the customer's own added cases.

In the combined weight tests there are also a few possible conclusions to take.

Comparing test 16 to test 17, it is clear that the numeric features properties, as currently implemented, are disruptive to the system's accuracy. They are meant to be a small factor to help the system distinguish similar cases, however, since the numeric data is not being normalized, its inclusion is separating the cases too much and outweighing the other parameters, making them almost irrelevant. Experimentation is still required to determine the best way to use this information, but a comparison between test 23 and test 24 proves that in some conditions these features can improve the system's results. Unlike the remaining tests, the values used in test 24 were adjusted to the test data to verify this hypothesis. Its result was the best of the group, with 97.37% optimal solutions provided, meaning it only failed one case. However, considering this configuration is tuned specifically for this testset and would not provide the same results in unrelated datasets, it is not correct to compare it to the remaining tests, for which the weight combinations were set *a priori*, and not incrementally adjusted to improve the results.

Tests 18 and 19 show that there are significant improvements when excluding the number of rows from the case comparisons. Other tests have been performed with several values below 1 and in none of these did the number of rows improve the results, which either remained unchanged or were worse upon enabling this information.

Excluding test 24, tests 20 and 23 had the best results achieved in this set, both selecting the desired chart type in 100% of the cases and doing so with a perfect use of attributes in 94,74% of them, resulting in a 98.82% overall score in the personalised metric. Considering the difference in their configuration, since test 23 achieves the same results as test 20 while using only 2 property weights, as opposed to 8, one might wonder if the remaining 6 weights are unnecessary. Upon closer inspection of both tests it is perceptible that the results from test 20 were provided with a higher level of knowledge of the problem than the results from test 23. For demonstration purposes, one of the test cases

(Appendix B) was individually ran in each of the configurations. The case-base ranking provided by each can be seen in Listing 4 and Listing 5.

```
[
  0.7142857142857143,   //case 0
  1.492063492063492,    //case 1
  0.7142857142857143,   //case 2
  0.7142857142857143,   //case 3
  1.492063492063492,    //case 4
  NaN,                  //case 5
  NaN,                  //case 6
  NaN,                  //case 7
  NaN,                  //case 8
  NaN,                  //case 9
  0.7142857142857143,   //case 10
  0                     //case 11
]
```

Listing 4 – Case-Base Ranking from Test 23

```
[
  2.8952380952380956,   //case 0
  6.492063492063492,    //case 1
  3.0380952380952384,   //case 2
  2.9892857142857143,   //case 3
  6.492063492063492,    //case 4
  NaN,                  //case 5
  NaN,                  //case 6
  NaN,                  //case 7
  NaN,                  //case 8
  NaN,                  //case 9
  2.9324675324675322,   //case 10
  0.8                   //case 11
]
```

Listing 5 – Case-Base Ranking from Test 20

Like with the remaining cases, both configurations provided the same result, which for this test was an adaptation of case 11's solution, the lowest scoring case (lower score means that the case is less different from the problem case). However, test 20 selected this option with a much better recognition of the different cases. In each example the most similar case is clear, but if the user provides negative feedback the system will need to evaluate the solution of the second most similar case. The configuration from test 20 determines that case 0 is the most similar from the remaining, followed by case 10, case 3 and case 2. The configuration from test 23 however, labels all these cases as being the same, and would select whichever had the lowest index. As of now, the number of tests and case diversity is limited, so it is possible that such unfounded selection would still produce the correct result, but it is clear that the most future-proof approach is the solution with the best case differentiator.

6.3.3 Final Remarks

Although the 94.74% of optimal answers achieved in tests 20 and 23 is not a perfect result, it should be remembered that in some cases determining the best visualization is a complex task even for the user. Some data might just not be good for visualizing, yet the system must be ready to output a visualization if it is required to.

It is also relevant to clarify that the accuracy results represent the system's ability to produce a solution in accordance to the cases it has been exposed to, i.e. the accuracy test routine is determining if the test passed or failed based on the solution described in each of the test cases. Yet, as discussed in Section 3.1.2, determining the best visualization for a given data is not an objective task. This means if a user were to use the system in its current state, he/she might dislike visualizations which the accuracy tests would consider correct. Likewise, what is currently considered an *optimal* answer is not necessarily the same output that is generated by the rule based system in use in Ansa. For example, for the query "show me my employees by title", the rule-based system outputs the bar chart shown in Figure 6.1. When this problem was introduced in the CBR case-base, it was defined that the most appropriate answer was instead the hoverable pie chart shown in

Figure 6.2. Whether one is better than the other is debatable but not relevant for these tests. Having this single case was enough for the system to match the expectation and choose pie charts over bar charts for similar data, which goes to show the learning and flexibility potential of the CBR system. Different users may prefer different solutions, and for each of those users, the system's case-base would be configured accordingly. If multiple users are using the same system, creating user profiles would be an option for dealing with this challenge in future iterations.

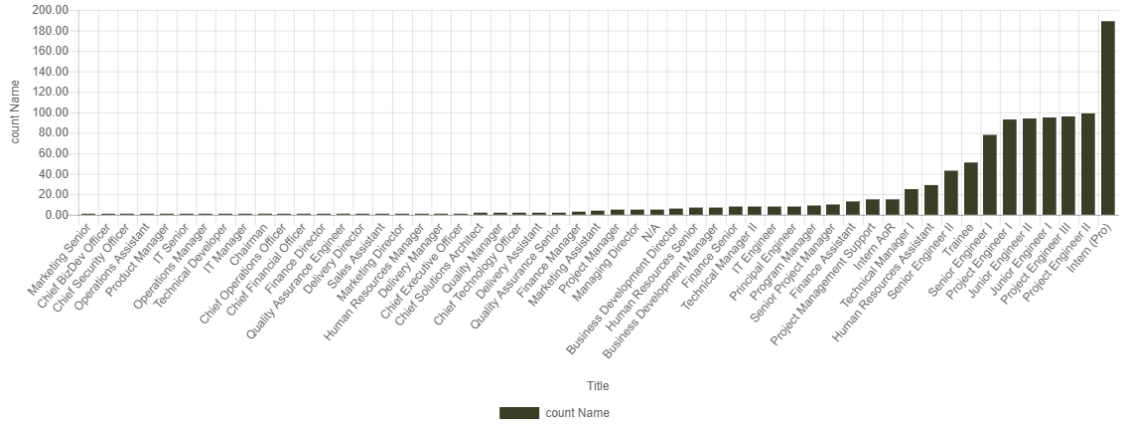


Figure 6.1 - RBS Proposed Chart for "Employees by Title" Query

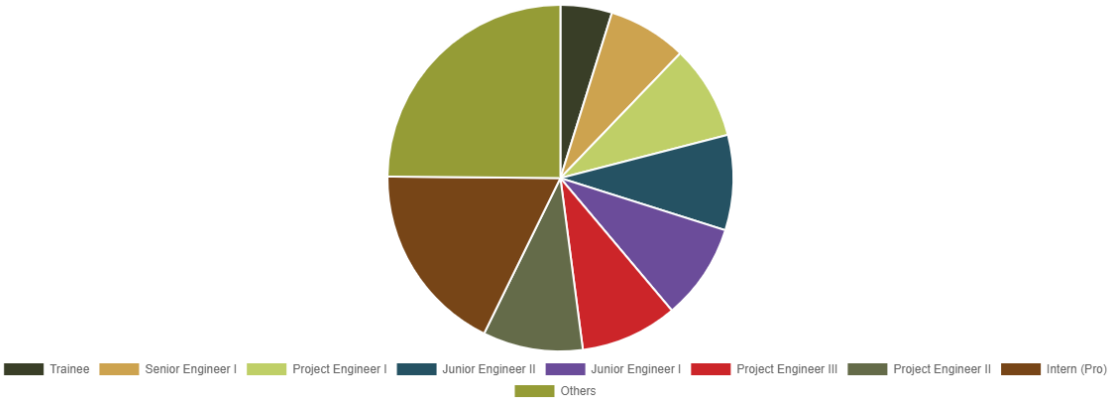


Figure 6.2 - CBR Proposed Chart for "Employees by Title" Query

In addition, a CBR system is constantly evolving. As it is used, it learns new cases, which changes its accuracy. Any case for which the system produced sub-optimal results in these tests, if added to the case-base would then belong to the knowledge base, and new problems similar to them would, ideally, not fail again.

Overall these experiments show promising results, indicating that using case-based reasoning is a valid possibility for Ansa. It is evident that the weights' tuning is a crucial part of the system's accuracy. While these accuracy and sensibility tests provided a great overview on which are the most relevant properties and combinations to consider, another interesting approach for future work would be to use genetic algorithms to determine the best possible combination across numerous testset and case-base combinations. It would also be very interesting to put this system to the test by adding new chart types. In the current RBS, adding a new chart type would require new rules for determining when it

would be viable, new rules for how to arrange the attributes and properly configure the chart to produce that representation, and the adjustment of previous rules to determine when the new option would be the preferred response. On the other hand, all the CBR system needs is reference cases and feedback.

Chapter 7

Conclusions

The system developed in this dissertation addresses the problem of deciding the most appropriate visual representation for data retrieved by Ansa to answer users' queries. Case-based reasoning, the selected approach for the challenge, was an appealing method for providing a solution that did not require expertise in the domain, extensive rule definition or complex maintenance.

Unlike the previous strategies, the implemented system is able to learn from introduced and experienced cases, resulting in a knowledge base in constant improvement and flexible to new graphical solutions which might come up in Ansa's future. It consists of three main interconnected modules, which together complete the case-based reasoning cycle. In the case-base ranking module, the system's known cases are ranked according to their similarity to the problem case, based on a similarity function with adjustable weights for each case feature. The solution generation module is responsible for reusing the known case most similar to the new problem to compose a solution for it. This solution includes the most appropriate chart type to use as well as how to use each of the data attributes, assigning them to their corresponding axis or defining if they should be used as the representation series. The feedback management module analyses the user feedback and manages the case-base accordingly or commands the generation of a different solution. Successful solutions are added to the case-base as new cases, stored in an easily readable format for easy maintenance if necessary.

Both the accuracy and performance tests have shown encouraging results. The system was able to exactly match the expected output in 94.74% of the test cases. In the remaining 5.26%, the solutions provided selected the correct chart type but arranged the axes in a sub-optimal way, yet not completely different from the desired organization (the correct Y axis was chosen, but the X axis and the series were swapped). Performance-wise, the system proved to be able to easily handle analysing hundreds of compatible cases with a near real-time response, well under the set performance requirement.

In conclusion, the developed CBR approach has shown to be a potential candidate for Ansa's system, showing great results, and it has encouraged the Ansa team to continue studying this possibility for their system. Naturally, this being the first implementation attempt of this technique, there is still room for improvements. These will be discussed in the following section. The undertaken work also provided the Ansa team with a first close experience with Vue.js, a powerful JavaScript framework being considering for use

in the system's frontend for its features and usability. It has shown to be a practical and accessible tool, providing intuitive mechanisms for instant interactions with the web app.

7.1 Future Work

In this section the current limitations of the system are addressed, while also reflecting on future areas of improvement and experimentation.

7.1.1 Current Improvements

Although the system has provided good results in its current form, there are some aspects to be perfected in future iterations.

As mentioned in the tests section, the numeric features included in the case representation are not currently improving CBR decisions. This happens because these features differ greatly from case to case, meaning that comparing their values, multiplying them by the corresponding weight and adding them directly to the case ranking completely outweighs the comparison results of the remaining properties. One option would be to collect this numeric information and normalize it in a common scale for all cases.

The comparison of nominal properties could also be improved. The similarity function compares nominal properties using the Levenshtein distance. While this method works for differentiating them and detecting which are the same, for the current implementation it would be better to replace it for a standard check verifying if the strings are equal or not.

Also able to be improved is the system's reaction when it is unable to provide the user with new solutions. As mentioned in Section 5.2.4, if the user evaluates a response with negative feedback, the system generates a different solution from its known cases. However, it is possible that the system runs out of different solutions to generate. In this case, it would be interesting to provide the user with the tools to generate the solution himself/herself, arranging the data in the most convenient way within the system's known visualization types. If the user was satisfied with his/her own visual representation, it could be directly added to the case-base.

It should also be noted that new solutions for cases included in the case-base are not being stored at the moment (cases with an exact match to one of the case-base cases). This limitation can be fixed by replacing the previous solution with the new one.

These would be the first details to address in a future iteration of the developed system, most of which would be relatively straightforward to implement. In the following section, some interesting thoughts for future experimentation are presented.

7.1.2 Experimentation

Throughout the development, an experiment was conducted using full case multipliers. The purpose of these multipliers, unlike the property weights discussed so far, was to represent the whole case, affecting the likelihood of it being selected as the best case for adaptation (meaning the similarity results of each case would be multiplied by its

corresponding multiplier). The concept was to store any new case with the multiplier equal to 1, and slightly adjust it whenever the user would provide feedback to a solution generated from it. This would result in less useful cases being selected less often and vice versa. The idea was abandoned as it would have other consequences (e.g., giving negative feedback to a solution adapted from a case would not only reduce the likelihood of that case being selected for the current problem, but also for any future problems, for which it could possibly provide the best solution). While this idea did not provide good results as it was implemented, it was still useful as a learning exercise to increase knowledge on CBR systems. The experimentation suggestions presented in this section may be equally profitable exercises, even if they do not accomplish their intent.

Currently the system is using a set of 14 properties to represent a case. While it is key to limit these properties to meaningful characteristics of the data, it would be interesting to experiment with supplementary ones and evaluate how these affect the system's performance, such as the number of distinct elements present in the attribute, or the information gain.

Other advances could be attempted by extracting information from the user's query. At the moment, the CBR process is focusing only in the data returned from the database (after back-end processing). However, knowing how the query was phrased could provide valuable information on how the user is expecting the answer to be displayed (e.g., by searching "show me my sales by country by product" the user might expect to see countries in the X-axis and products as series, while by searching "show me my sales by product by country" the opposite might be expected).

Another possible experiment wold be to implement a negative case-base. Whenever a proposed solution received negative feedback it could be added to the negative case-base along with the corresponding problem, building a collection of known "bad solutions". After calculating the case-base ranking, the system would update it accordingly by comparing each solution to the negative case-base, or possibly only comparing the best solution to make sure it is not very similar to a known negative case. Other ways of managing these extra cases could also be attempted. This technique is mentioned in CBR theory [25][58], but there is not much documentation on systems actually using it, although some work is currently under development [59].

It is also worth it to analyse the way feedback and case retention is implemented. Currently, the system adapts when the user provides criticism. However, perhaps the absence of feedback could be taken as an indication of a positive result. On the same topic, studies should be made on the most effective way to handle new learned cases. Should all learned cases be added to the case-base? If yes, over time the knowledge base will get increasingly bigger, which could lead to worse performance and case-base saturation. Performance issues could be counteracted by using a different organization strategy (see Section 3.3.3), but the increasing amount of cases would result in each case being less meaningful (e.g., if the user adds a new case with a different solution than the typical one for a particular kind of problem, this new case might have no effect on the following problems, as the chance of being selected would be much lower than in a reduced case-base, with tighter case selection). If not all successful cases are added, it is necessary to determine how to select which cases should or should not be added to the knowledge base.

In the long run, user profiles are also a possibility. These would allow for a more personalized experience for each user, but they would slow down the growth of the

knowledge base. It would be important to find the right balance between common cases available to all users and personalized cases available only to the user who generated them.

7.1.3 Ansa

Ansa is a promising project with a great team backing it up, for whom this section presents two visualization-related suggestions.

As B. Witzen [7, p. 3] puts it: *"Besides the imagery itself, being able to interact with the visualization, thereby providing the end-user with the means to dynamically adjust the visualization, has also grown in appreciation".* In the future it would be excellent to experiment with new kinds of charts and charting mechanisms. Chart.js, the chart engine in use, does provide some other interesting visualization options, such as stacked charts, scatter charts, bubble charts, various scale options and more [60], although it does not provide tools for users to interact with the visualizations. This natural step forward would broaden the complexity of the domain, making more use of the case-based reasoning system, which could also increase the density of its responses, including extra parameters for chart design, such as scales, styles, positioning, 3D options, etc.

Lastly, it seems like one extra valuable feature for Ansa would be the inclusion of a general dashboard, with an overview of the main available data. An instrument such as Ansa, which allows the user to ask questions about his/her data and get insightful visual responses, is a powerful tool. However, for some users the problem might just be knowing which questions to ask. Independently of the user's experience with the system, having a set of automatic visualizations easily available, while also displaying Ansa's different charting capabilities, may open the door for new analysis ideas. This dashboard, customizable by the user, could be in the web app's main page, which currently contains only the search bar.

References

[1] D. a Keim, “Information visualization and visual data mining,” *IEEE Trans Vis Comput Graph*, vol. 8, no. 1, pp. 1–8, 2002.

[2] J. D. Mackinlay, P. Hanrahan, and C. Stolte, “Show Me: Automatic presentation for visual analysis,” *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1137–1144, 2007.

[3] Y. Lee and M. Negnevtsky, “Rule-based Expert Systems,” in *CS560 Knowledge Discovery and Management*, 2004.

[4] Becerra-Fernandez, “Chapter 9: Using Past History Explicitly as Knowledge: Case-Based Reasoning Systems.” Prentice Hall, 2008.

[5] M. O. Manuel Sánchez-Montañés, Julia Díaz, “Advantages and Disadvantages of Rule-based Reasoning.” Escuela Politécnica Superior, UAM, pp. 3–6.

[6] R. Mazza, *Introduction to Information Visualization*. Springer, 2009.

[7] B. Witzen, “Automatic data visualization,” University of Amsterdam, 2014.

[8] S. John Walker, “Review of ‘Big Data: A Revolution That Will Transform How We Live, Work, and Think’ by Viktor Mayer-Schönberger and Kenneth Cukier,” *Norwich University of the Arts*. 2014.

[9] D. Cunningham, “Analysis of the geometrical patterns found in the lascaux cave system,” *Midnight Sci.*, 2014.

[10] W. Litte, *Shorter Oxford English Dictionary*. Oxford University Press, 1972.

[11] “Visualization,” *English Oxford Dictionaries*, 2018. [Online]. Available: https://en.oxforddictionaries.com/definition/visualization. [Accessed: 27-May-2018].

[12] C. Ware, *Information Visualization: Perception for Design: Second Edition*. Morgan Kaufmann Publishers, 2004.

[13] E. R. Tufte, “The visual display of quantitative information.” Graphics Press, 1985.

[14] L. A. Cary Jensen, *Harvard graphics 3: the complete reference*. Osborne McGraw-Hill, 1992.

[15] C. E. Woods, *Organizing a factory*. The System Company, 1905.

[16] H. Wainer, *Visual revelations: graphical tales of fate and deception from Napoleon Bonaparte to Ross Perot*. Lawrence Erlbaum Associates, Inc., 1997.

[17] Lærd Statistics, "Understanding Histograms." [Online]. Available: https://statistics.laerd.com/statistical-guides/understanding-histograms.php. [Accessed: 23-Apr-2018].

[18] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110–141, 1986.

[19] Y. Sun, J. Leigh, A. Johnson, and S. Lee, "Articulate: A semi-automated model for translating natural language queries into meaningful visualizations," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6133 LNCS, pp. 184–195, 2010.

[20] T. . Gao, M. . Dontcheva, E. . Adar, Z. . Liu, and K. . Karahalios, "Datatone: Managing ambiguity in natural language interfaces for data visualization," *Proc. 28th Annu. ACM Symp. User Interface Softw. Technol. - UIST '15*, pp. 489–500, 2015.

[21] A. Srinivasan and J. Stasko, "NL4DV : Toolkit for Natural Language Driven Data Visualization," *IEEE Vis. Poster Proc.*, 2016.

[22] M. X. Zhou and M. Chen, "Automated generation of graphic sketches by example," *IJCAI Int. Jt. Conf. Artif. Intell.*, no. Figure 1, pp. 65–71, 2003.

[23] B. Freyne, Jill; Smyth, "Creating Visualizations: A Case-Based Reasoning Perspective," in *AICS'09 Proceedings of the 20th Irish conference on Artificial intelligence and cognitive science*, 2009, pp. 82–91.

[24] A. Aamodt and E. Plaza, "Case-Based Reasoning : Foundational Issues , Methodological Variations , and System Approaches," *AI Commun.*, vol. 7, no. 1, pp. 39–59, 1994.

[25] J. L. Kolodner, "An introduction to case-based reasoning," *Artif. Intell. Rev.*, vol. 6, no. 1, pp. 3–34, 1992.

[26] A. J. Gonzalez and R. Laureano-Ortiz, "A case-based reasoning approach to real estate property appraisal," *Expert Syst. Appl.*, vol. 4, no. 2, pp. 229–246, 1992.

[27] M. Pantic, "Introduction to Machine Learning & Case-Based Reasoning," *Introduction to Machine Learning*, no. course 395. Imperial College London, p. 20, 2012.

[28] B. Smyth and P. Cunningham, "Hierarchical Case-Based Reasoning Techniques for Plant-Control Software Design Hierarchical Case-Based Reasoning," *Computer (Long. Beach. Calif).*

[29] T. Steffens, *Enhancing Similarity Measures with Imperfect Rule-based Background Knowledge*. Ios PressInc, 2006.

[30] I. Watson and F. Marir, "Case-based reasoning: A review," *Knowl. Eng. Rev.*, vol. 9, no. 04, p. 327, Dec. 1994.

[31] R. Bareiss, *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning*. 1989.

[32] D. Leake, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*.

AAAI Press, 1996.

[33] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2001.

[34] K. Schwaber, "SCRUM Development Process," *Springer-Verlag London Ltd. 1997*, no. February 1986, pp. 1994–1995, 1997.

[35] Stigasoft, "Agile Methodology," 2018. [Online]. Available: http://www.stigasoft.com/agile-scrum-methodology.html. [Accessed: 22-Jun-2018].

[36] "Introducing GitFlow," *DataSift*. [Online]. Available: https://datasift.github.io/gitflow/IntroducingGitFlow.html. [Accessed: 25-May-2018].

[37] "Issue Management Tools - Popularity Ranking," *Project Management Zone*, 2018. [Online]. Available: https://project-management.zone/ranking/category/issue. [Accessed: 25-May-2018].

[38] P. Bourque and R. E. Fairley, *Guide to the Software Engineering - Body of Knowledge.* IEEE Computer Society, 2014.

[39] US Department of Defense Systems Management College, *Systems Engineering Fundamentals*, Defence Ac., no. January. 2001.

[40] R. B. Miller, "Response time in man-computer conversational transactions," *Proc. December 9-11, 1968, fall Jt. Comput. Conf. part I - AFIPS '68 (Fall, part I)*, p. 267, 1968.

[41] J. Nielsen, "Usability Engineering," *Morgan Kaufmann Pietquin O Beaufort R*, vol. 44, no. 1/2002, p. 362, 1993.

[42] T. Huston, "What is Microservices Architecture?," *SmartBear*, 2018. [Online]. Available: https://smartbear.com/learn/api-design/what-are-microservices/. [Accessed: 26-May-2018].

[43] L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps," *IEEE Int. Conf. Softw. Archit. (ICSA 2018)*, 2018.

[44] StackOverflow, "Developer Survey Results 2017." [Online]. Available: https://insights.stackoverflow.com/survey/2017#technology-programming-languages. [Accessed: 26-May-2018].

[45] W3Techs, "Usage of JavaScript for websites," 2018. [Online]. Available: https://w3techs.com/technologies/details/cp-javascript/all/all. [Accessed: 26-May-2018].

[46] S. Cass, "The 2017 Top Programming Languages," *IEEE Spectrum*, 2017. [Online]. Available: https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages. [Accessed: 26-May-2017].

[47] S. Misirlakis, "The 7 Most In-Demand Programming Languages of 2018," 2017. [Online]. Available: https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/. [Accessed: 26-May-2018].

[48] A. Petkov, "Best Programming Languages to Learn in 2018," 2018. [Online]. Available: https://medium.freecodecamp.org/best-programming-languages-to-learn-in-2018-ultimate-guide-bfc93e615b35.

[49] P. Staudacher, “Top Programming Languages to Learn on the IBM i for 2018,” *TALSCO*, 2018. [Online]. Available: https://www.talscoinc.com/top-programming-languages-to-learn-on-the-ibm-i-for-2018/. [Accessed: 26-May-2018].

[50] DA-14, “5 Best JavaScript Frameworks in 2017,” 2017. [Online]. Available: https://da-14.com/blog/5-best-javascript-frameworks-2017. [Accessed: 26-May-2018].

[51] M. Fowler, “Inversion Of Control,” 2005. [Online]. Available: https://martinfowler.com/bliki/InversionOfControl.html. [Accessed: 26-May-2018].

[52] S. Kumar, “Difference Between Library and Framework,” *C#Corner*, 2015. [Online]. Available: https://www.c-sharpcorner.com/UploadFile/a85b23/framework-vs-library/. [Accessed: 26-May-2018].

[53] F. Acet, “The Growth of Vue.js is Phenomenal,” *Medium*, 2018. [Online]. Available: https://medium.com/@amsterdamjs/fatih-acet-the-growth-of-vue-js-is-phenomenal-5875337c1f6. [Accessed: 26-May-2018].

[54] J. S. Park, “The status of JavaScript libraries & frameworks: 2018 & beyond,” *Medium*, 2018. [Online]. Available: https://medium.com/@alberto.park/the-status-of-javascript-libraries-frameworks-2018-beyond-3a5a7cae7513. [Accessed: 04-Jun-2018].

[55] P. C. Rica, “What is Vue.js and What are its Advantages,” *Hackernoon*, 2017. [Online]. Available: https://hackernoon.com/what-is-vue-js-and-what-are-its-advantages-4071b7c7993d. [Accessed: 26-May-2018].

[56] Vuejs.org, “Using Axios to Consume APIs,” 2018. [Online]. Available: https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html. [Accessed: 27-May-2018].

[57] G. Michael, “Levenshtein Distance, in Three Flavors,” *Merriam Park Software*. [Online]. Available: http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein Distance.htm. [Accessed: 24-May-2018].

[58] M. M. Richter and R. O. Weber, *Case-Based Reasoning: A Textbook*. Springer, 2013.

[59] E. Nauer and L. Lieber, “PhD proposal: Reasoning with positive and negative cases,” *University of Lorraine*, 2018. [Online]. Available: http://www.loria.fr/fr/emplois/thesis-offer-reasoning-with-positive-and-negative-cases/. [Accessed: 18-Jun-2018].

[60] Chart.js, “Chart.js Samples,” *Chartjs.org*, 2018. [Online]. Available: http://www.chartjs.org/samples/latest/. [Accessed: 20-Jun-2018].

Appendix

Appendix A – Resultset from the query “2016 revenue by month”


```
{
      "httpCode": 200,
      "data": {
        "ansaQuery": {
          "queryWindow": {"limit": -1, "offset": 0},
          "selected": [{
            "underlyingColumn": {
              "transform": "MONTH",
              "attribute": {"id": 5, "attributeName": "Date", "type": "TIMESTAMP"},
              "type": "TRANSFORMED_ATTRIBUTE",
              "valueType": "NUMERIC",
              "name": "month Date"
            }, "valueType": "NOMINAL", "type": "CAST", "name": "month Date"
          }, {
            "transform": "SUM",
            "attribute": {"id": 9, "attributeName": "Value", "type": "NUMERIC"},
            "type": "TRANSFORMED_ATTRIBUTE",
            "valueType": "NUMERIC",
            "name": "sum Value"
          }],
          "groupingColumns": [{
            "underlyingColumn": {
              "transform": "MONTH",
              "attribute": {"id": 5, "attributeName": "Date", "type": "TIMESTAMP"},
              "type": "TRANSFORMED_ATTRIBUTE",
              "valueType": "NUMERIC",
              "name": "month Date"
            }, "valueType": "NOMINAL", "type": "CAST", "name": "month Date"
          }],
          "orderingColumns": [{
            "underlyingColumn": {
              "transform": "MONTH",
              "attribute": {"id": 5, "attributeName": "Date", "type": "TIMESTAMP"},
              "type": "TRANSFORMED_ATTRIBUTE",
              "valueType": "NUMERIC",
              "name": "month Date"
            }, "valueType": "NOMINAL", "type": "CAST", "name": "month Date"
          }, {
            "transform": "SUM",
            "attribute": {"id": 9, "attributeName": "Value", "type": "NUMERIC"},
            "type": "TRANSFORMED_ATTRIBUTE",
            "valueType": "NUMERIC",
            "name": "sum Value"
          }],
          "tables": [{"id": 3, "name": "Sales", "type": "PLAIN"}],
          "ordering": "ASCENDING",
          "restriction": {
            "predicateOperator": "EQ",
            "column": {
              "transform": "YEAR",
              "attribute": {"id": 5, "attributeName": "Date", "type": "TIMESTAMP"},
              "type": "TRANSFORMED_ATTRIBUTE",
              "valueType": "NUMERIC",
              "name": "year Date"
            },
            "predicateBinding": {"value": {"type": "NUMERIC", "value": 2016.0}, "type": "VALUE"},
            "type": "PREDICATE"
          },
          "distinct": false,
          "uuid": "03e11b6f-86a0-4fa2-8920-3899a36e01f7"
        },
        "dataViews": [{
          "dataViewType": "TABLE_VIEW",
          "valid": true,
          "resultSetHeaderItems": [{"id": 0, "text": "month Date", "attributeType": "NOMINAL"}, {
            "id": 1,
            "text": "sum Value",
            "attributeType": "NUMERIC"
          }],
          "rows": [["1", 1323966.0], ["2", 881607.0], ["3", 1114628.0], ["4", 957496.0], ["5", 862956.0],
["6", 1549919.0], ["7", 1535122.0], ["8", 1478494.0], ["9", 1271930.0], ["10", 1370998.0]]
        }]
      },
      "version": "2.1",
      "requestDuration": 50
    }
```

Appendix B – “Employees by unit” Resultset


```
{
      "httpCode": 200,
      "data": {
        "ansaQuery": {
          "queryWindow": {"limit": -1, "offset": 0},
          "selected": [{
            "transform": "COUNT",
            "attribute": {"id": 10, "attributeName": "Name", "type": "NOMINAL"},
            "type": "TRANSFORMED_ATTRIBUTE",
            "valueType": "NUMERIC",
            "name": "count Name"
          }, {
            "attribute": {"id": 7, "attributeName": "Unit", "type": "NOMINAL"},
            "type": "ATTRIBUTE",
            "valueType": "NOMINAL",
            "name": "Unit"
          }],
          "groupingColumns": [{
            "attribute": {"id": 7, "attributeName": "Unit", "type": "NOMINAL"},
            "type": "ATTRIBUTE",
            "valueType": "NOMINAL",
            "name": "Unit"
          }],
          "orderingColumns": [{
            "transform": "COUNT",
            "attribute": {"id": 10, "attributeName": "Name", "type": "NOMINAL"},
            "type": "TRANSFORMED_ATTRIBUTE",
            "valueType": "NUMERIC",
            "name": "count Name"
          }],
          "tables": [{"id": 3, "name": "Employees", "type": "PLAIN"}],
          "ordering": "ASCENDING",
          "restriction": null,
          "distinct": false,
          "uuid": "e45ce357-70cf-4957-b35b-5ffc13e12f0c"
        },
        "dataViews": [{
          "dataViewType": "TABLE_VIEW",
          "valid": true,
          "resultSetHeaderItems": [{"id": 0, "text": "count Name", "attributeType": "NUMERIC"}, {
            "id": 1,
            "text": "Unit",
            "attributeType": "NOMINAL"
          }],
          "rows": [[10.0, "TWT"], [11.0, "Big Bang"], [13.0, "hyperCRITICAL"], [21.0, "Stormtroopers"], [25.0,
"Vision"], [27.0, "Patinhas"], [36.0, "Romulans"], [41.0, "Vulcan"], [45.0, "Klingon"], [46.0, "Rebels"]]
        }, {
          "dataViewType": "PIE_CHART_VIEW",
          "valid": true,
          "slices": [{"sliceName": "N/A", "value": 393.0}, {
            "sliceName": "The Expendables",
            "value": 108.0
          }, {"sliceName": "Machimbombo", "value": 67.0}, {
            "sliceName": "Skywalkers",
            "value": 57.0
          }, {"sliceName": "Borg", "value": 50.0}, {
            "sliceName": "Rebels",
            "value": 46.0
          }, {"sliceName": "Delivery Office", "value": 46.0}, {
            "sliceName": "Klingon",
            "value": 45.0
          }, {"sliceName": "Others", "value": 184.0}],
          "pieChartMapper": {"VALUE_INDEX": 0, "LABEL_INDEX": 1}
        }, {
          "dataViewType": "BAR_CHART_VIEW",
          "valid": true,
          "axisMap": {"Y_AXIS": 0, "X_AXIS": 1},
          "series": [{
            "name": "count Name",
            "dataList": [["TWT", 10.0], ["Big Bang", 11.0], ["hyperCRITICAL", 13.0], ["Stormtroopers", 21.0],
["Vision", 25.0], ["Patinhas", 27.0], ["Romulans", 36.0], ["Vulcan", 41.0], ["Klingon", 45.0], ["Rebels",
46.0], ["Delivery Office", 46.0], ["Borg", 50.0], ["Skywalkers", 57.0], ["Machimbombo", 67.0], ["The Expenda-
bles", 108.0], ["N/A", 393.0]]
          }]
        }]
      },
      "version": "2.1",
      "requestDuration": 23
    }
```

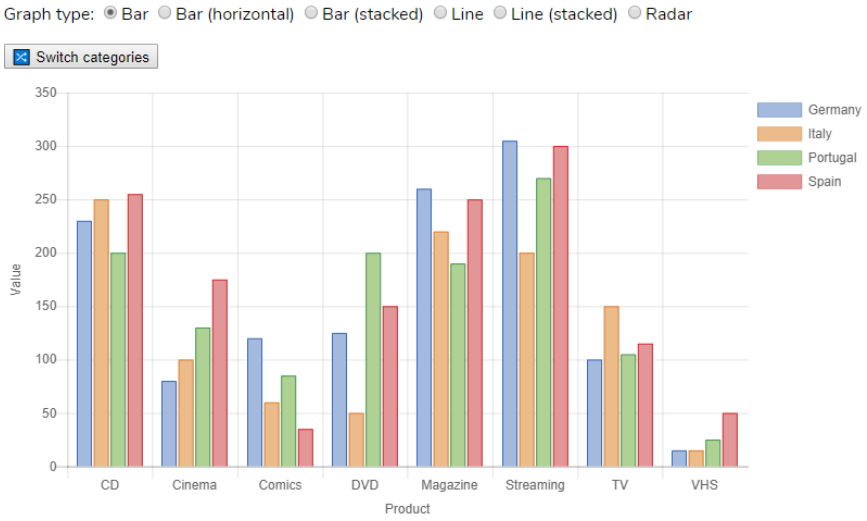
Appendix C – "Volume by country by quarter" Case in JSON

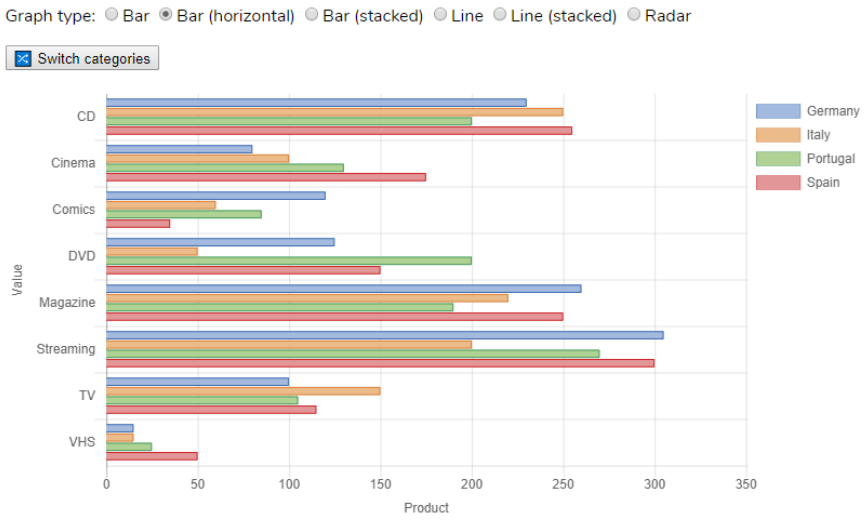

```
{
    "generalFeatures": {
      "nCols": 3,
      "nRows": 10,
      "nDateCol": 1,
      "nNominalCol": 1,
      "nNumericCol": 1
    },
    "attributes": [
      {
        "id": 0,
        "transform": "SUM",
        "valueType": "NUMERIC",
        "attributeName": "Volume",
        "attributeType": "NUMERIC",
        "max": 756,
        "min": 406,
        "mean": 624.7,
        "median": 692.5,
        "stdev": 128.07501707983488
      },
      {
        "id": 1,
        "transform": "None",
        "valueType": "NOMINAL",
        "attributeName": "Country",
        "attributeType": "NOMINAL",
        "max": 0,
        "min": 0,
        "mean": 0,
        "median": 0,
        "stdev": 0
      },
      {
        "id": 2,
        "transform": "QUARTER_TRUNC",
        "valueType": "TIMESTAMP",
        "attributeName": "Date",
        "attributeType": "TIMESTAMP",
        "max": 0,
        "min": 0,
        "mean": 0,
        "median": 0,
        "stdev": 0
      }
    ],
    "solution": {
      "dataViewType": "PLOT_VIEW",
      "axisMap": {"SERIES": 1, "X_AXIS": 2, "Y_AXIS": 0},
    }
```

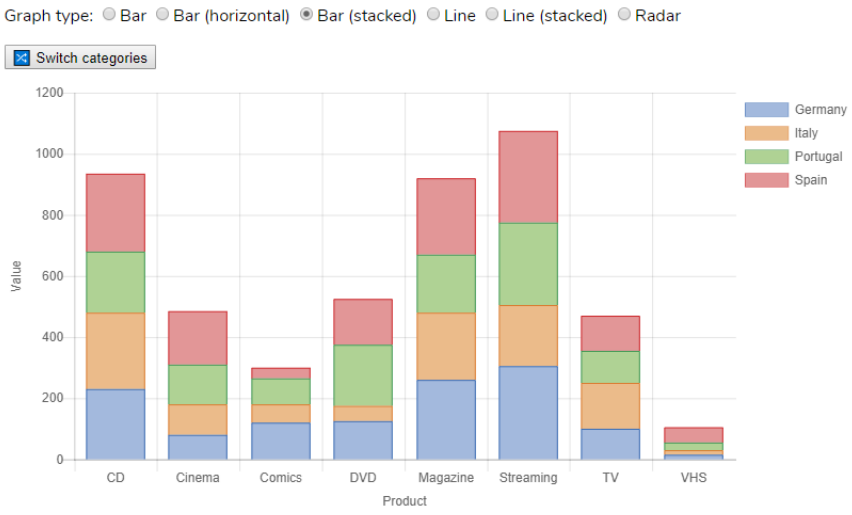
Appendix D – Simplified Array Representation of the "Volume by country by quarter" Case

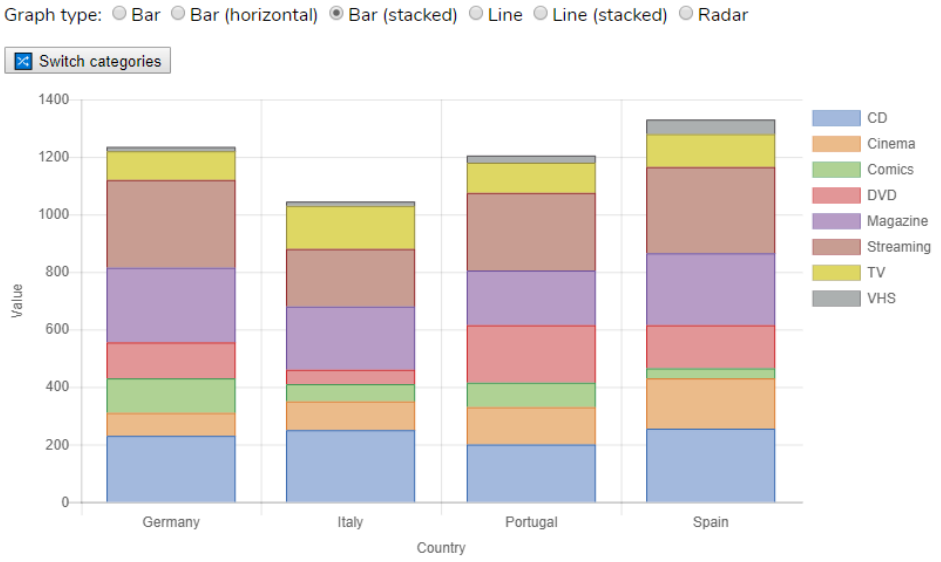

```
[
  [
    3,
    10,
    1,
    1,
    1
  ],
  [
    [
      0,
      "SUM",
      "NUMERIC",
      "Volume",
      "NUMERIC",
      756,
      406,
      624.7,
      692.5,
      128.07501707983488
    ],
    [
      1,
      "None",
      "NOMINAL",
      "Country",
      "NOMINAL",
      0,
      0,
      0,
      0,
      0
    ],
    [
      2,
      "QUARTER_TRUNC",
      "TIMESTAMP",
      "Date",
      "TIMESTAMP",
      0,
      0,
      0,
      0,
      0
    ]
  ],
  [
      "PLOT_VIEW",
      [
        1, 2, 0
      ],
  ]
]
```

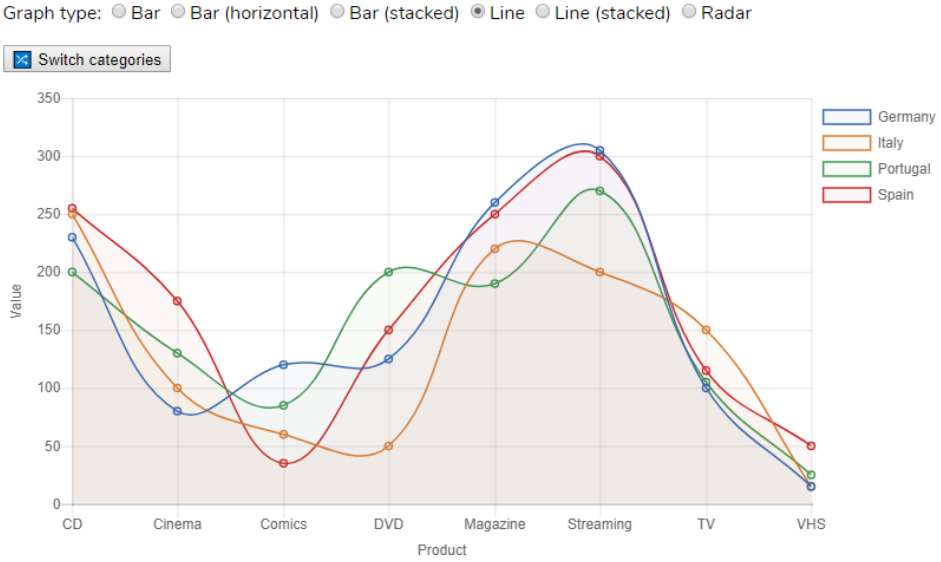
Appendix E - Visualizations Created with Chart.js

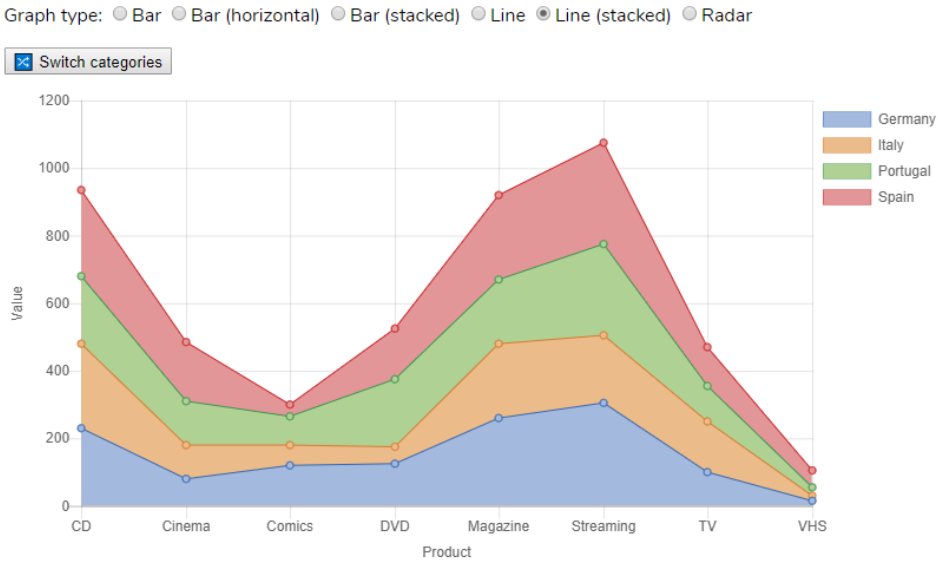


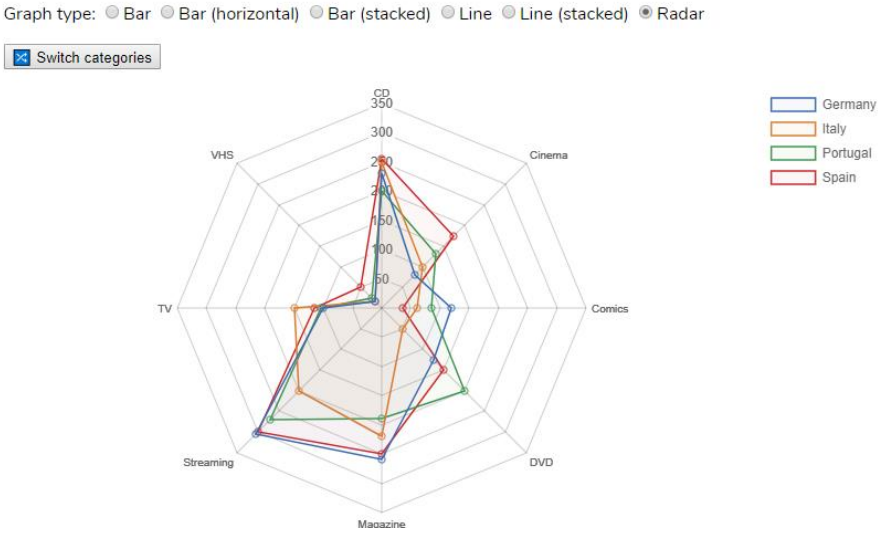


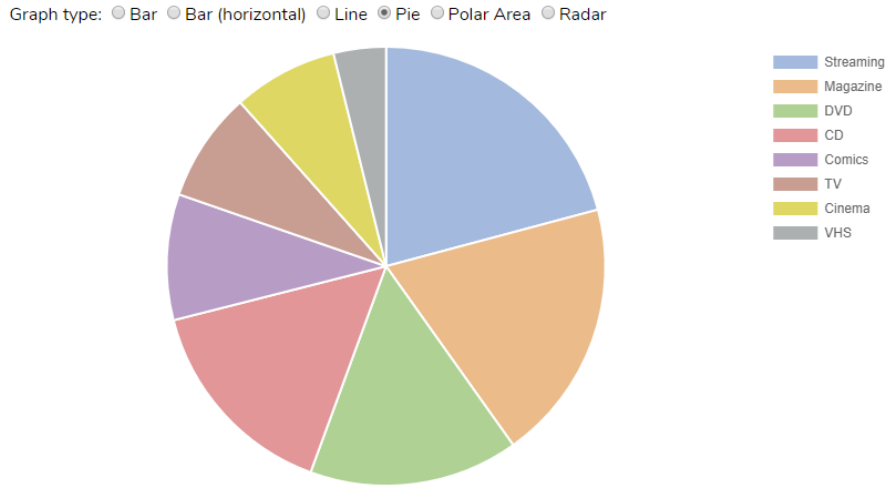


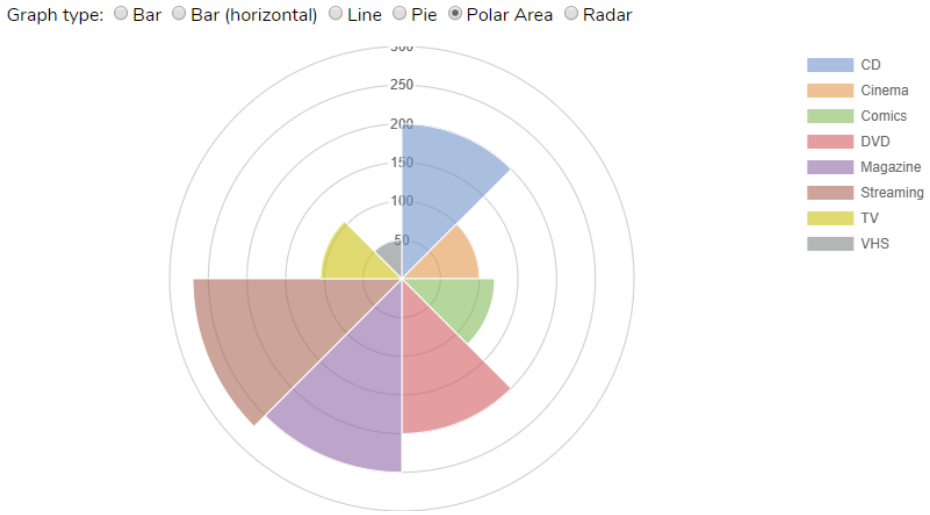

Graph type: Bar Bar (horizontal) Bar (stacked) Line Line (stacked) Radar
Switch categories
1400
1200
1000
800
600
400
200
0
Value
Germany
Italy
Portugal
Spain
Country
CD
Cinema
Comics
DVD
Magazine
Streaming
TV
VHS


Graph type: Bar Bar (horizontal) Bar (stacked) Line Line (stacked) Radar
Switch categories
350
300
250
200
150
100
50
0
Value
CD
Cinema
Comics
DVD
Magazine
Streaming
TV
VHS
Product
Germany
Italy
Portugal
Spain


Graph type: Bar Bar (horizontal) Bar (stacked) Line Line (stacked) Radar
Switch categories
1200
1000
800
600
400
200
0
Value
CD
Cinema
Comics
DVD
Magazine
Streaming
TV
VHS
Product
Germany
Italy
Portugal
Spain


Graph type: Bar Bar (horizontal) Bar (stacked) Line Line (stacked) Radar
Switch categories
CD
Cinema
Comics
DVD
Magazine
Streaming
TV
VHS
Germany
Italy
Portugal
Spain


Graph type: Bar Bar (horizontal) Line Pie Polar Area Radar
Streaming
Magazine
DVD
CD
Comics
TV
Cinema
VHS


Graph type: Bar Bar (horizontal) Line Pie Polar Area Radar
250
200
150
100
50
CD
Cinema
Comics
DVD
Magazine
Streaming
TV
VHS